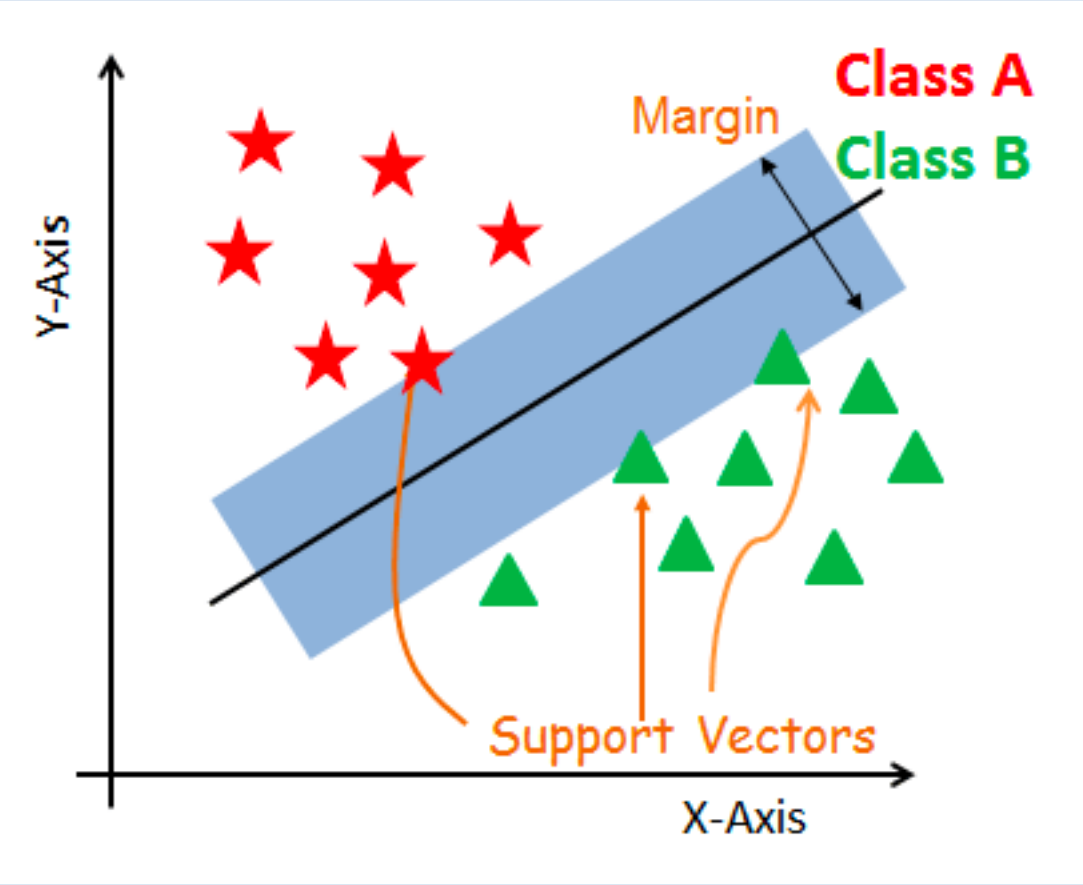
Applied Machine Learning
Week 6:
Classification:
SVM, Decision Trees, Random Forest, KNN and Combining ML Techniques

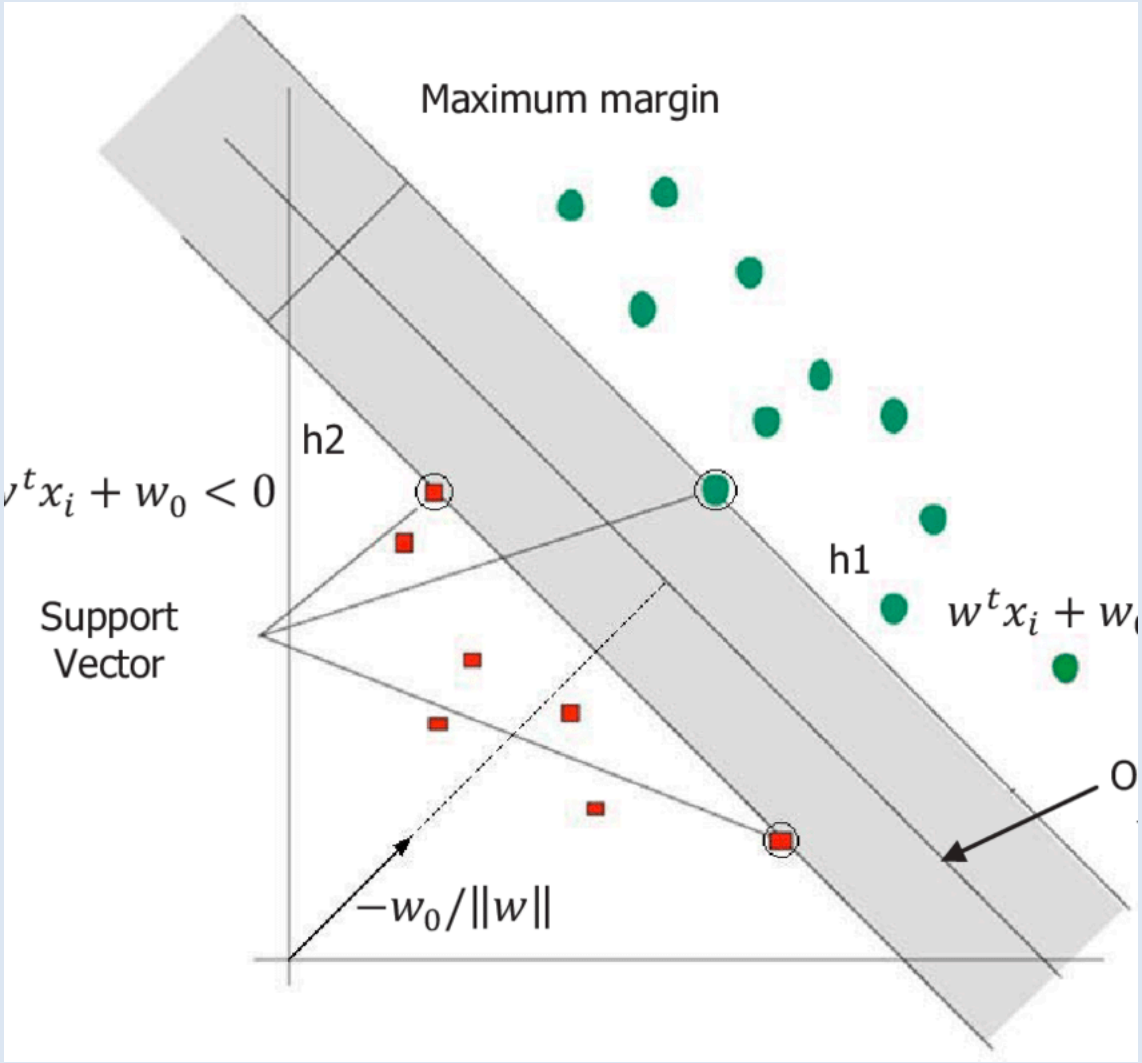
Dr. Ali Arsanjani

Support Vector Machines

- Generally, Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems.
- It can easily handle multiple continuous and categorical variables.
- SVM constructs a hyperplane in multidimensional space to separate different classes.
- SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error.
  - Today Stochastic Gradient Descent to minimize an error
    - y_correct – y_hat_predicted_value
- The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.

SVM





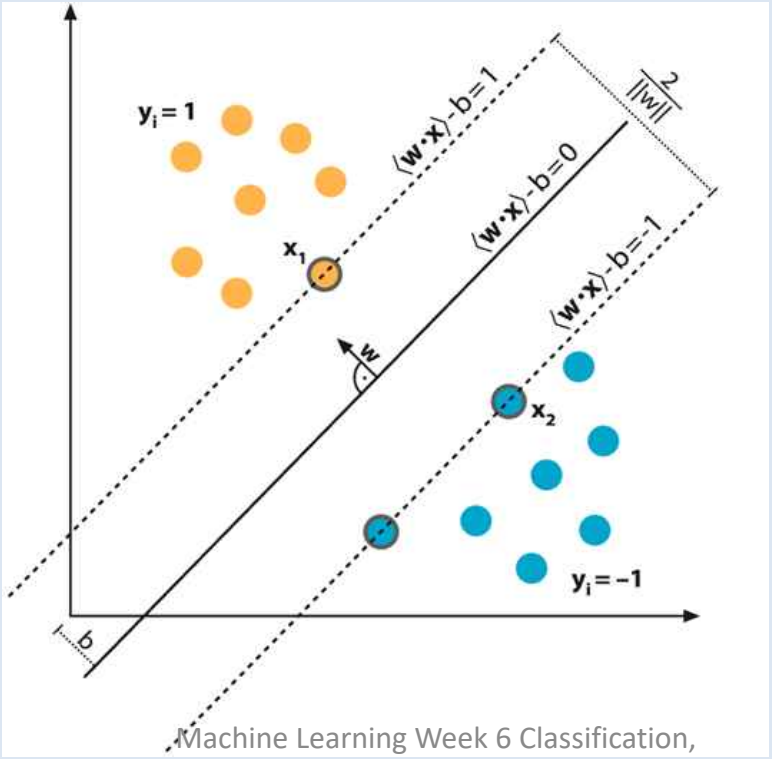
https://www.researchgate.net/figure/Maximum-margin-hyperplane-and-margins-for-an-SVM_fig4_284188649

SVM Terms

- **Support Vectors**
  - Support vectors are the data points, which are closest to the hyperplane.
  - define the separating line better by calculating margins.
  - Points that are more relevant to the construction of the classifier.
- **Hyperplane**
  - A hyperplane is a decision plane which separates between a set of objects having different class memberships.
- **Margin**
  - A margin is a gap between the two lines on the closest class points.
  - calculated as perpendicular distance from the line to support vectors or closest points.
  - If the margin is larger in between the classes, then it is considered a good margin, a smaller margin is a bad margin.

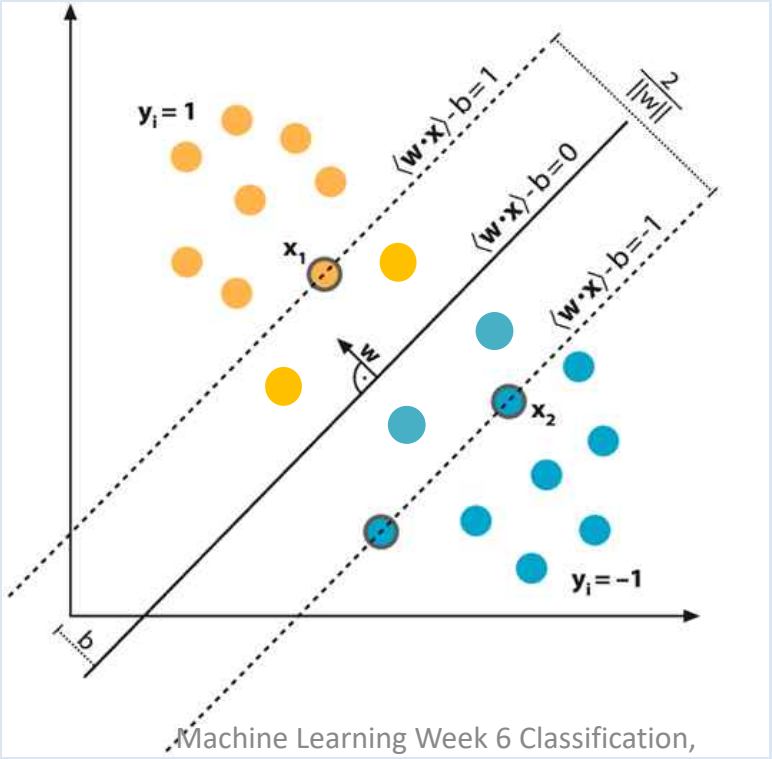
Support Vector Machines

- A Support Vector Machine (SVM) is a classifier that tries to **maximize the margin** between training data and the classification boundary (the plane defined by $X\beta = 0$)



Support Vector Machines

- The idea is that maximizing the margin **maximizes the chance that classification will be correct on new data**. We assume the new data of each class is near the training data of that type.



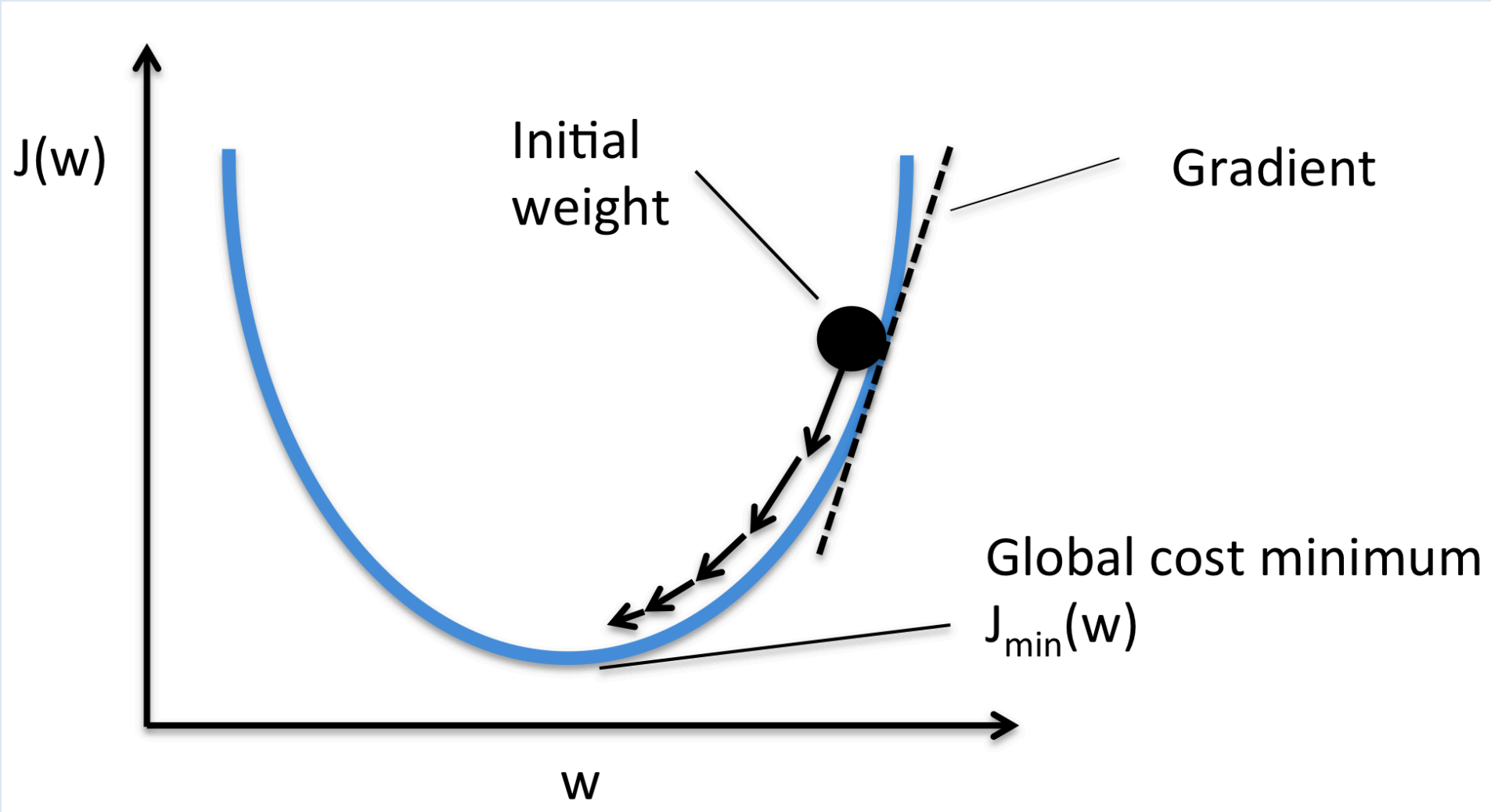
SVM Training

SVMs can be trained using SGD, so can deep neural networks
(Perceptron, Multi-layer perceptron (MLP) == fully connected network)

The **SVM gradient can be defined** as (here $p^i = X^i\beta$)

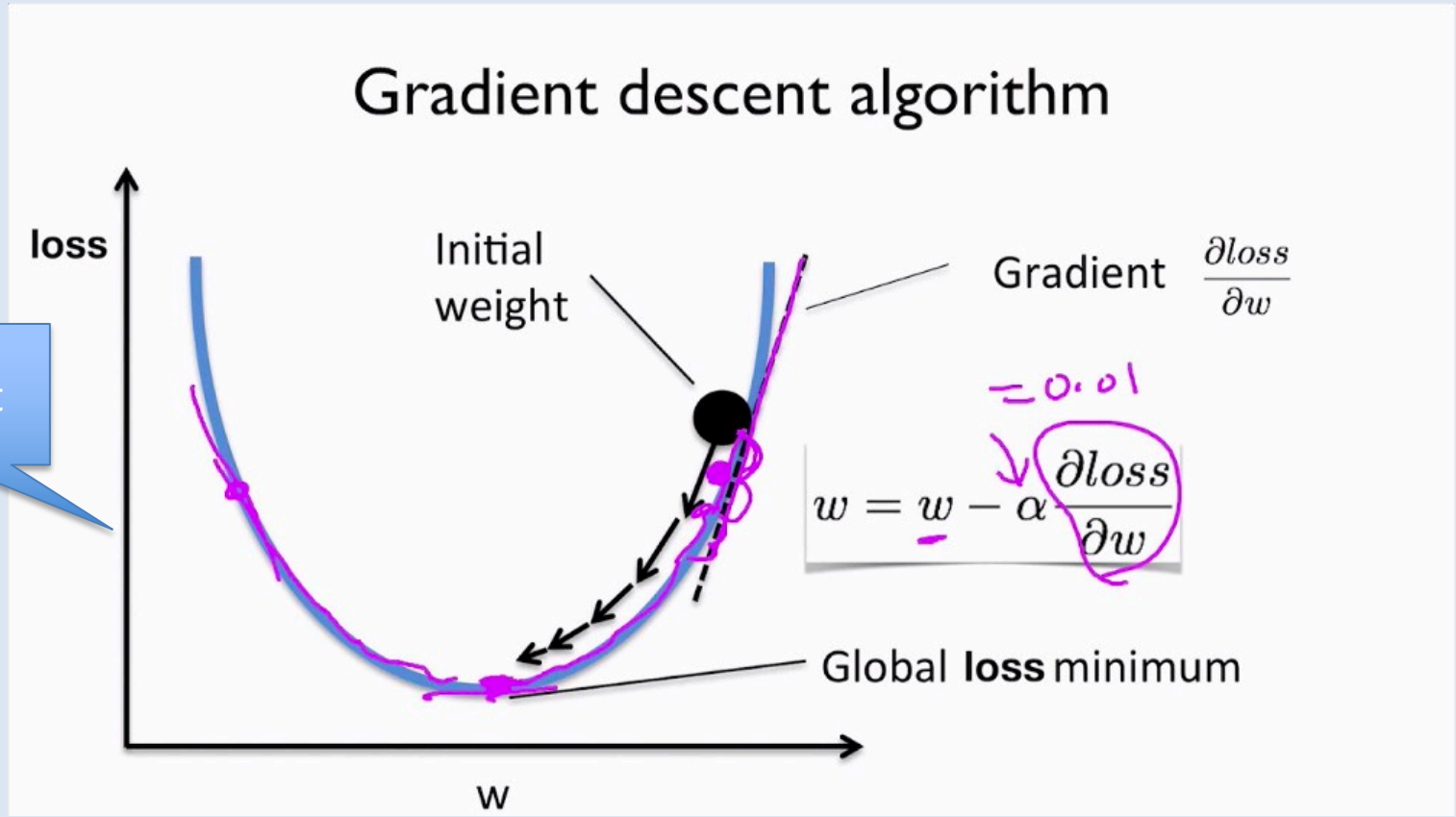
$$\frac{dA}{d\beta} = \sum_{i=1}^{N} \text{if } \left(p^i y^i < 1\right) \text{ then } y^i X^i \text{ else } 0$$

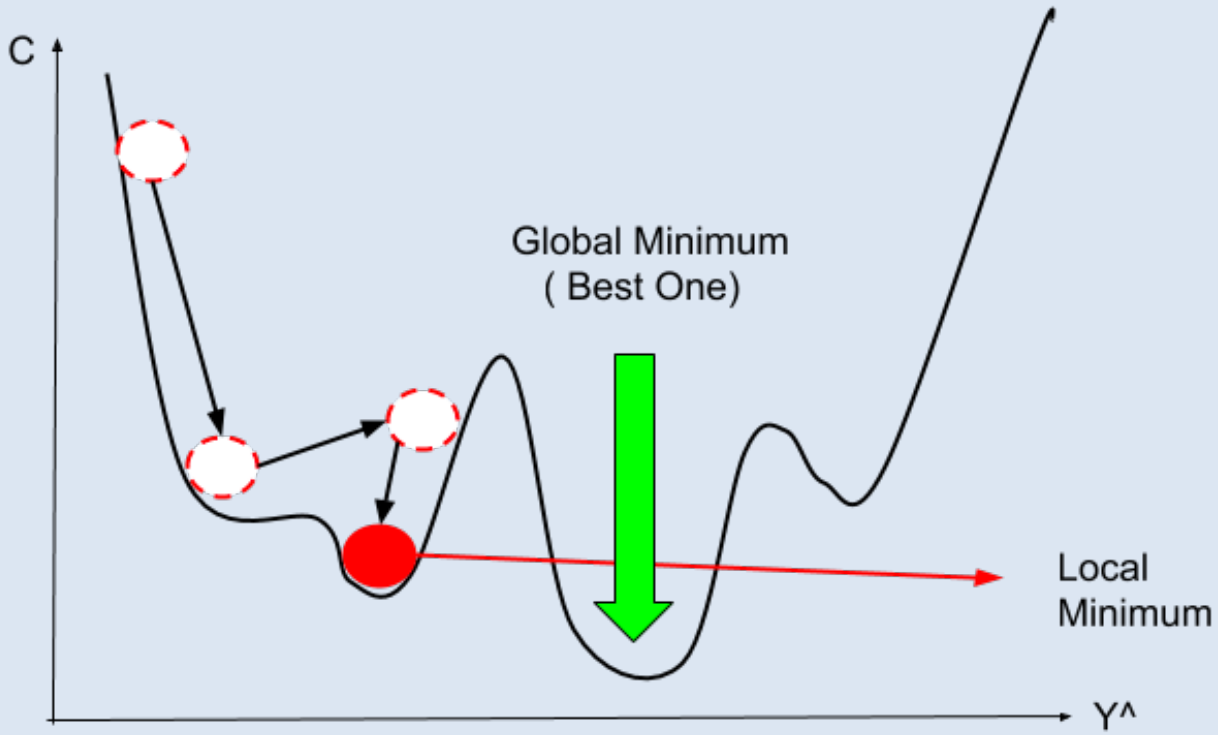
The expression $\left(p^i y^i < 1\right)$ tests whether the point X^i is nearer than the margin, and if so adds it with sign y^i. This "nudges" the model to push it further out next time. It ignores other points.


J(w)
Initial
weight
Gradient
Global cost minimum
$J_{min}(w)$
w

dy/dx

Y - yhat

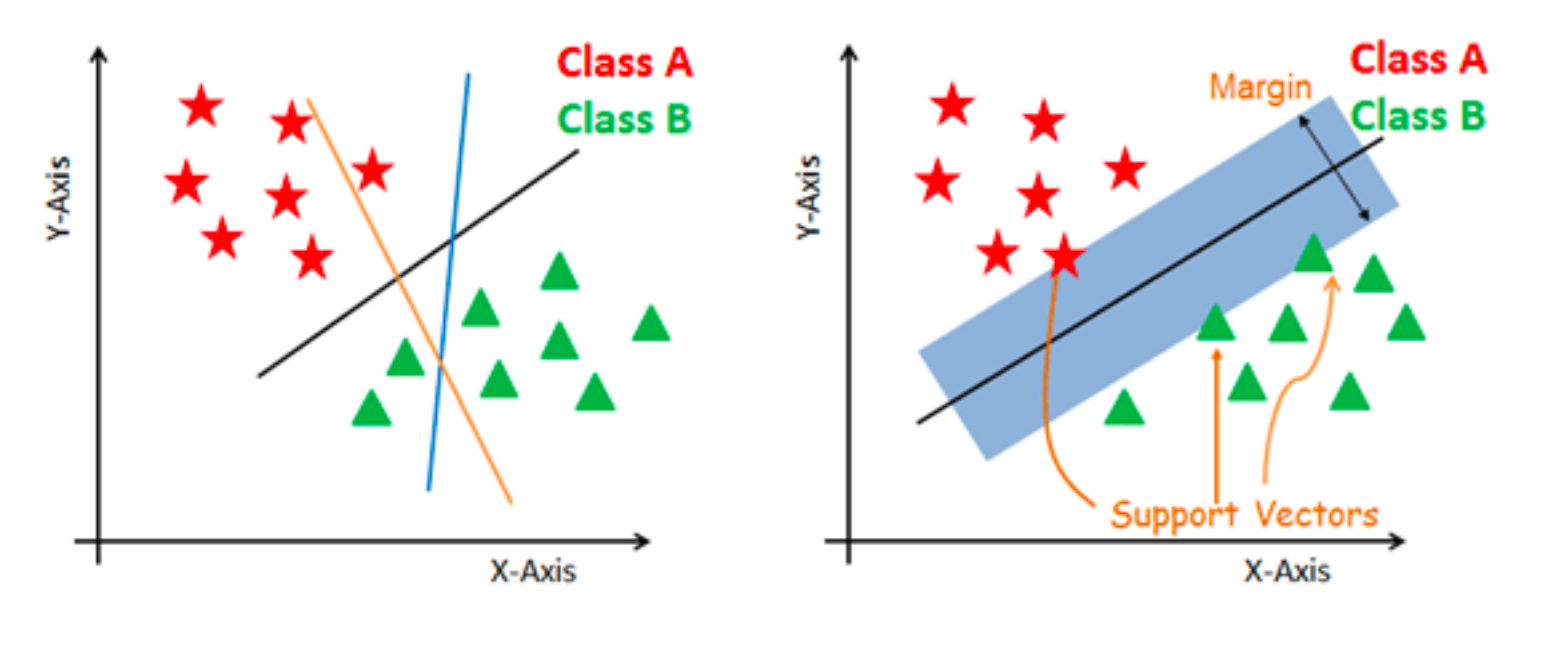

Gradient descent algorithm
loss
Initial weight
Gradient $\frac{\partial loss}{\partial w}$
$w = w - \alpha \frac{\partial loss}{\partial w}$
Global loss minimum
w


C
Global Minimum
(Best One)
Local
Minimum
Y^

SVM Heuristics

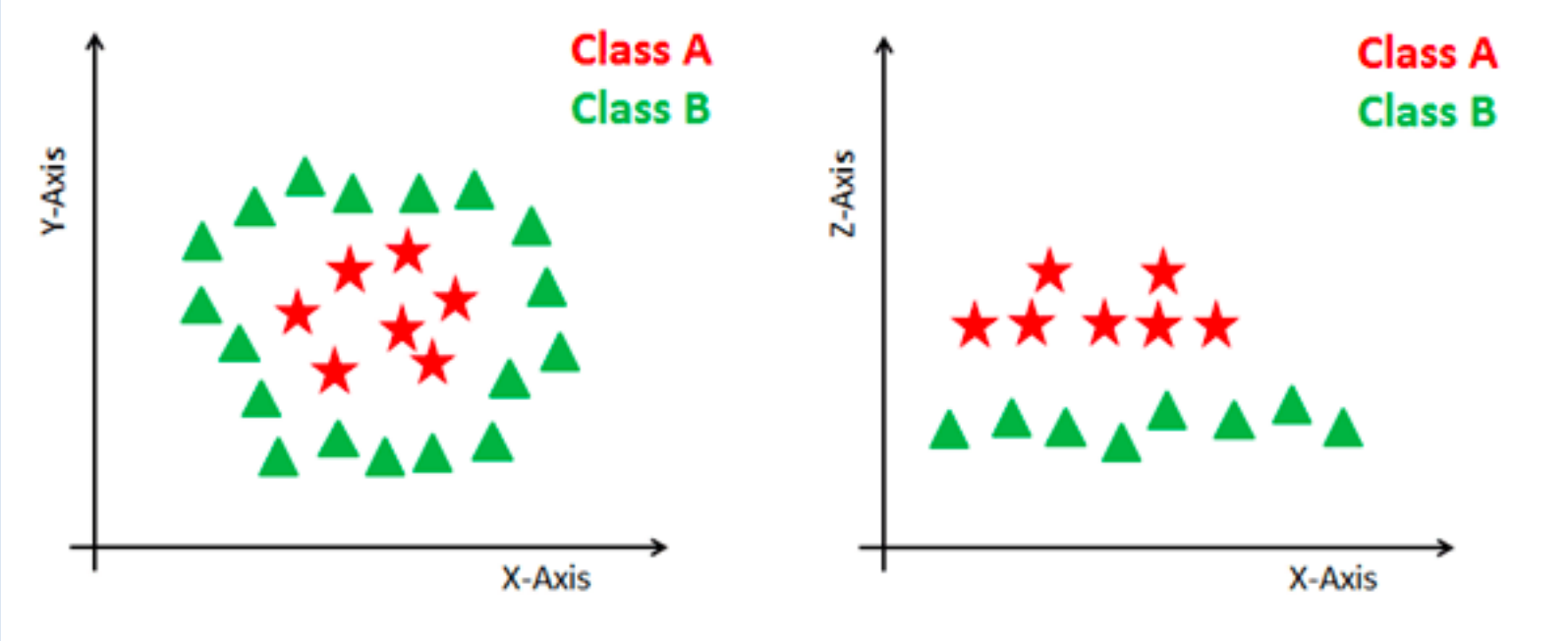
- The main objective is to segregate the given dataset in the best possible way.
- The distance between the either nearest points is known as the margin.
- The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset.
- SVM searches for the maximum marginal hyperplane in the following steps:
- Generate hyperplanes which segregates the classes in the best way.

SVM Heuristics

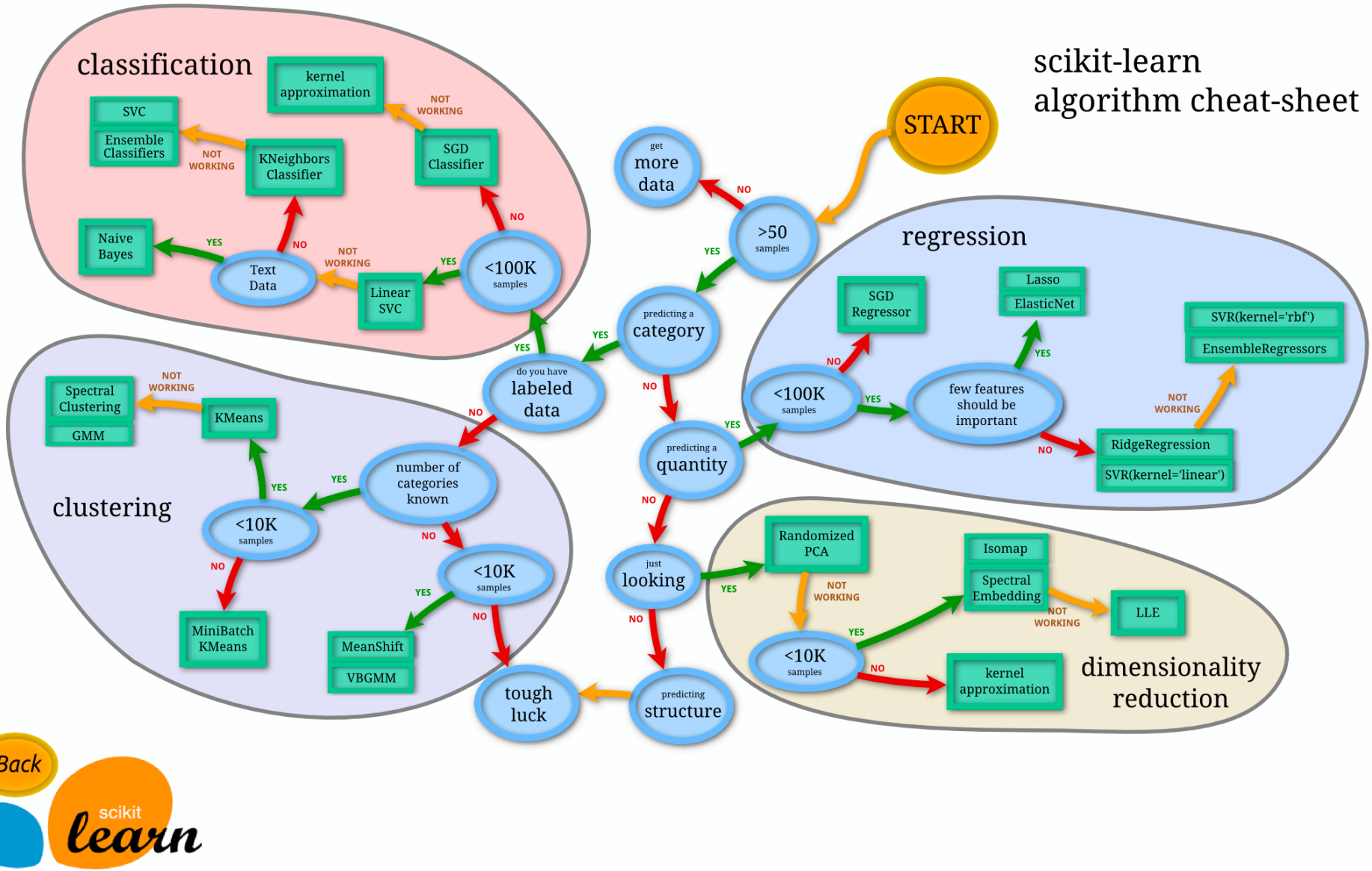


- Left-hand side figure showing three hyperplanes black, blue and orange.
- Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.
- Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.

Dealing with non-linear and inseparable planes



- Some problems can't be solved using linear hyperplane, as shown in the figure below (left-hand side).
- In such situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right.
- The data points are plotted on the x-axis and z-axis (Z is the squared sum of both x and y: z=x^2+y^2). Now you can easily segregate these points using linear separation.


scikit-learn
algorithm cheat-sheet
START
get more data
NO
>50 samples
YES
predicting a category
YES
NO
do you have labeled data
YES
NO
predicting a quantity
YES
NO
just looking
YES
NO
predicting structure
tough luck
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
Text Data
NO
NOT WORKING
Linear SVC
YES
<100K samples
YES
clustering
Spectral Clustering
GMM
NOT WORKING
KMeans
YES
number of categories known
YES
<10K samples
NO
MiniBatch KMeans
NO
<10K samples
YES
MeanShift
VBGMM
NO
regression
SGD Regressor
NO
<100K samples
YES
few features should be important
YES
Lasso
ElasticNet
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
dimensionality reduction
Randomized PCA
YES
NOT WORKING
<10K samples
YES
Isomap
Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
Back
scikit learn

Pattern = Muller Loop

```
classifiers = [
    KNeighborsClassifier(2),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
#     GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]
```

Add classifiers to this list:
XGBoost

Homework

Run all classifiers, find the best

```
Classifier = Nearest Neighbors, Score (test, accuracy) = 60.22 Training time = 47.40 seconds
Classifier = Linear SVM, Score (test, accuracy) = 63.39 Training time = 226.39 seconds
Classifier = RBF SVM, Score (test, accuracy) = 61.49 Training time = 368.53 seconds
Classifier = Decision Tree, Score (test, accuracy) = 59.44 Training time = 5.04 seconds
Classifier = Random Forest, Score (test, accuracy) = 57.06 Training time = 0.08 seconds
Classifier = Neural Net, Score (test, accuracy) = 67.00 Training time = 19.37 seconds
Classifier = AdaBoost, Score (test, accuracy) = 61.27 Training time = 58.89 seconds
Classifier = Naive Bayes, Score (test, accuracy) = 56.50 Training time = 0.19 seconds
Classifier = QDA, Score (test, accuracy) = 60.60 Training time = 2.20 seconds
--------------------------------------------------------------------------------
Best --> Classifier = Neural Net, Score (test, accuracy) = 67.00
```

```
]: dump(clf_best, 'clf_best.joblib')
```

Save the model, Pattern = Pickle and Load

https://www.journaldev.com/15638/python-pickle-example

```
import pickle

# take user input to take the amount of data
number_of_data = int(input('Enter the number of data : '))
data = []

# take input of the data
for i in range(number_of_data):
    raw = input('Enter data '+str(i)+' : ')
    data.append(raw)

# open a file, where you ant to store the data
file = open('important', 'wb')

# dump information to that file
pickle.dump(data, file)

# close the file
file.close()
```

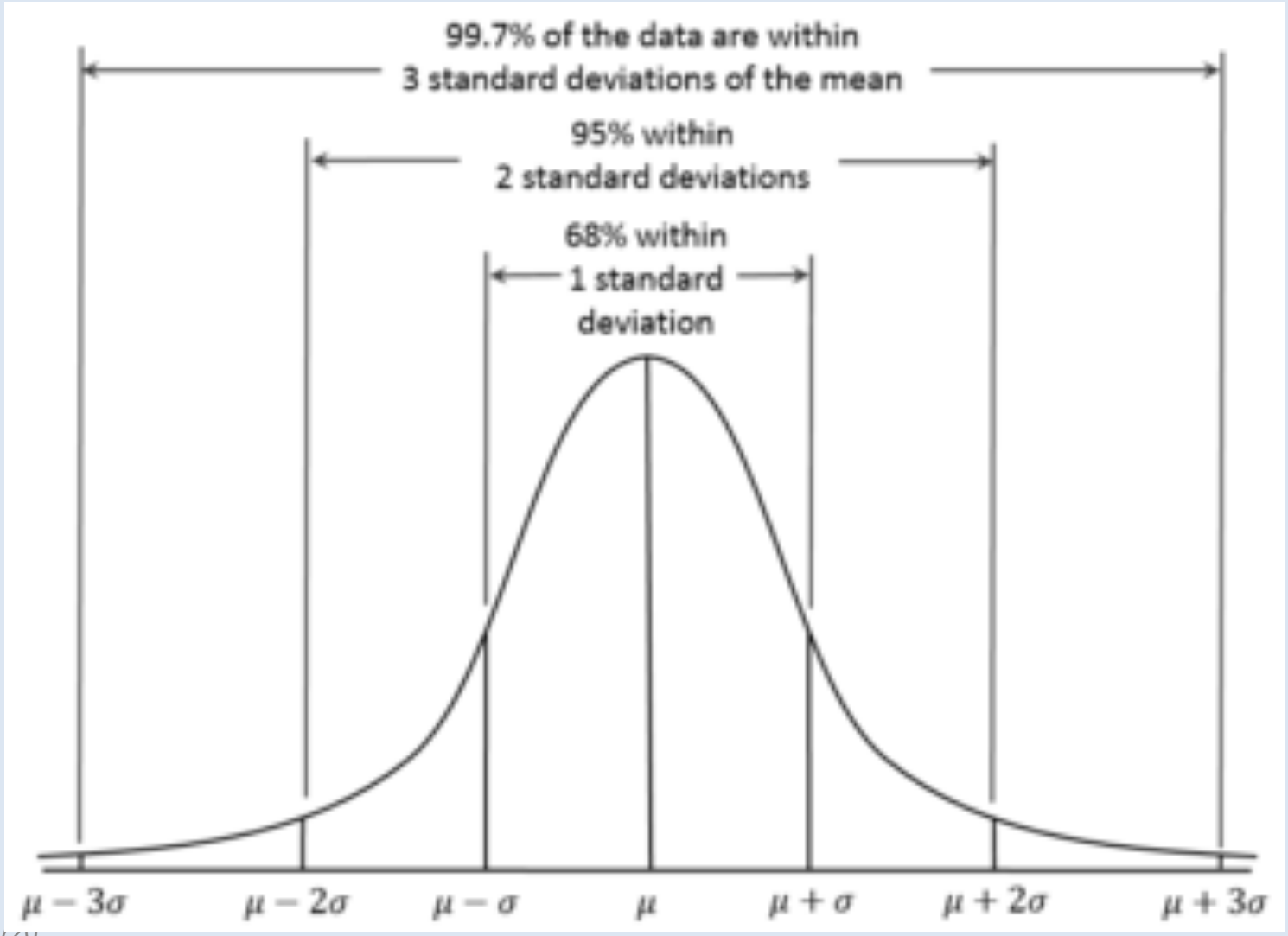
model.predict()

```
import pickle

# open a file, where you stored the pickled data
file = open('important', 'rb')

# dump information to that file
data = pickle.load(file)
```

model

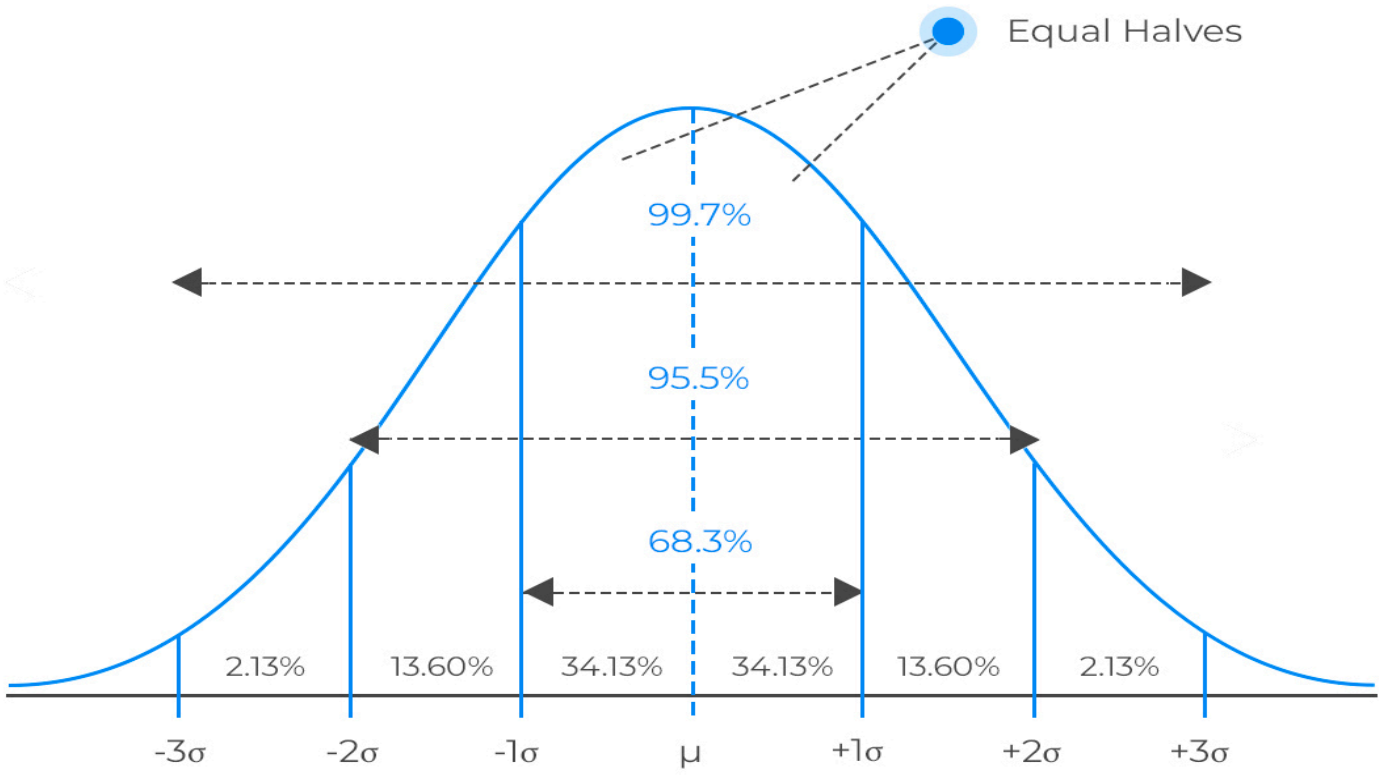
```
# close the file
file.close()

print('Showing the pickled data:')

cnt = 0
for item in data:
    print('The data ', cnt, ' is : ', item)
    cnt += 1
```

https://www.leansixsigmadefinition.com/glossary/gaussian-distribution/



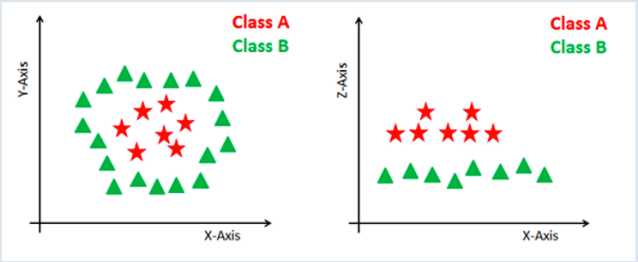
https://analystprep.com/cfa-level-1-exam/quantitative-methods/key-properties-normal-distribution/

Shape of the normal distribution



Dimension Transformations

- PCA
  - N → n
- SVM applies PCA to do the
  - Kernel trick
    - n → N
    - To make the data Linearly separable
    - RBF (radial Basis Function : z = y^2 + x^2)



RBF

- https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html
- sklearn.gaussian_process.kernels.RBF¶

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.gaussian_process import GaussianProcessClassifier
>>> from sklearn.gaussian_process.kernels import RBF
>>> X, y = load_iris(return_X_y=True)
>>> kernel = 1.0 * RBF(1.0)
>>> gpc = GaussianProcessClassifier(kernel=kernel,
...         random_state=0).fit(X, y)
>>> gpc.score(X, y)
0.9866...
>>> gpc.predict_proba(X[:2,:])
array([[0.8354..., 0.03228..., 0.1322...],
       [0.7906..., 0.0652..., 0.1441...]])
```

Stochastic Gradient

A very important set of iterative algorithms use **stochastic gradient** updates.

They use a **small subset or mini-batch X** of the data, and use it to compute a gradient which is added to the model

$$\beta' = \beta + \alpha \nabla$$

Where α is called the **learning rate**.

These updates happen **many times** in one pass over the dataset.

Its possible to compute high-quality models with very few passes, sometime with less than one pass over a large dataset.

Challenges for Stochastic Gradient

Stochastic gradient has some serious limitations however, especially if the **gradients vary widely in magnitude**. Some coefficients change very fast, others very slowly. (The Hessian corrects for this in Newton's method).

This happens for **text, user activity and social media data** (and other power-law data), because gradient magnitudes scale with feature frequency, i.e. over several orders of magnitude.

Its not possible to set a single learning rate that trains the frequent and infrequent features at the same time.

Creating Correct and Capable Classifiers

- A set of examples for diagnosing "correct and capable classifiers"
- pandas_profiling
- Starting with a baseline model before building 'better' classifiers
- Diagnosing which parts of model-space are easier to predict correctly
- YellowBrick for ROC and Confusion Matrices
- Confusion Matrix probabilities
- T-SNE to group together similar examples
- ELI5 and SHAPley to explain predictions

https://github.com/ianozsvald/data_science_delivered/blob/master/ml_creating_correct_capable_classifiers.ipynb

Separation of Dataset for Training

- Dataset
  - Instead of training the whole set at once
    - Mini-batches
      - Mini batch sgd
- Partition the dataset
  - Train one model
  - Then train another model
  - Then combine the two into a single outcome: ensemble model
- Begins to address the issue learnablity and explainability

Train test split

- Basic train test split
- Use the DummyClassifier
- Then use cv=RepeatedKFold(n_repeats=CV_N_REPEATS)
- K-fold cross validation

Black box Unexplainable Model

- Deep Learning World

Explaining the results of a classification

Traditional AI

Tagged Multiple-models in an ensemble

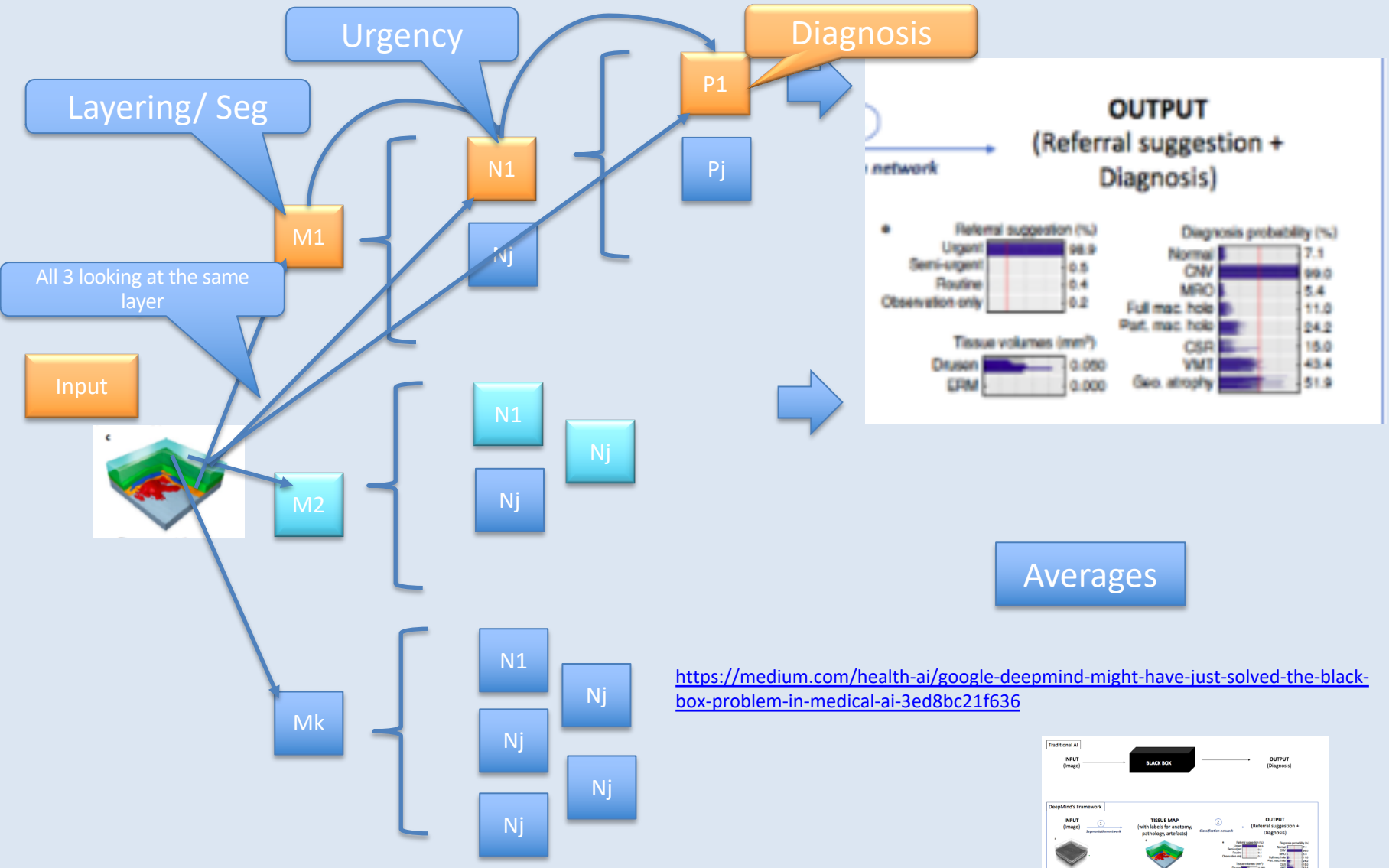
OUTPUT
(Diagnosis)

DeepMind's Framework

DeepMind's framework tackles the "Black Box" Problem by having 2 neural networks with a readily viewable ap) in between

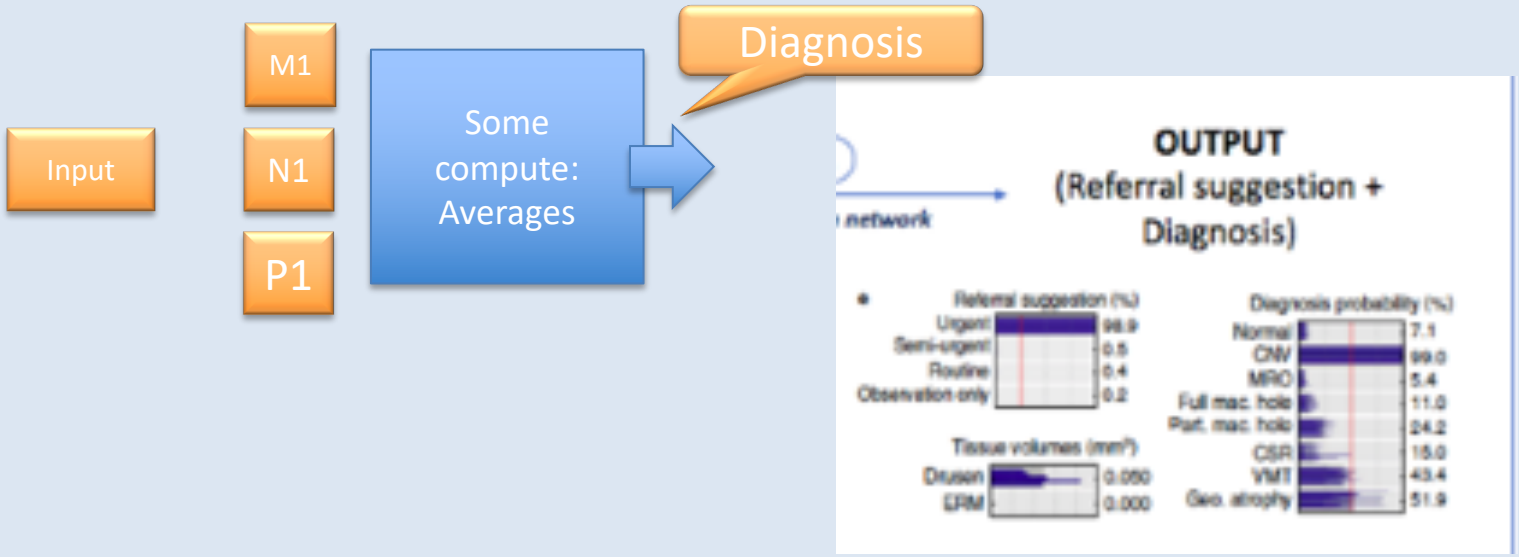
SHAP Values

Stacked (tagged) Multi-model Classification



https://medium.com/health-ai/google-deepmind-might-have-just-solved-the-black-box-problem-in-medical-ai-3ed8bc21f636

Ensemble : Voting among simpler or weaker models



ADAGRAD – Adaptive-rate SGD

ADAGRAD is a particularly simple and fast approach to this problem. The gradient g_t at the t[th] step is scaled as

$$\frac{g_t}{\sqrt{\sum_{k=1}^{t} {g_k}^2}}$$

Where ${g_k}^2$ is the k[th] gradient vector squared element-wise.

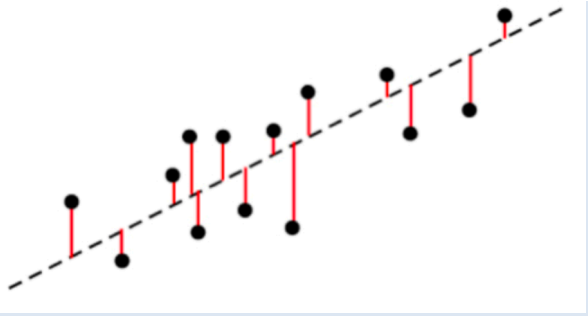
This corrects for feature scale factors, and all ADAGRAD-scaled gradient components have similar magnitudes.

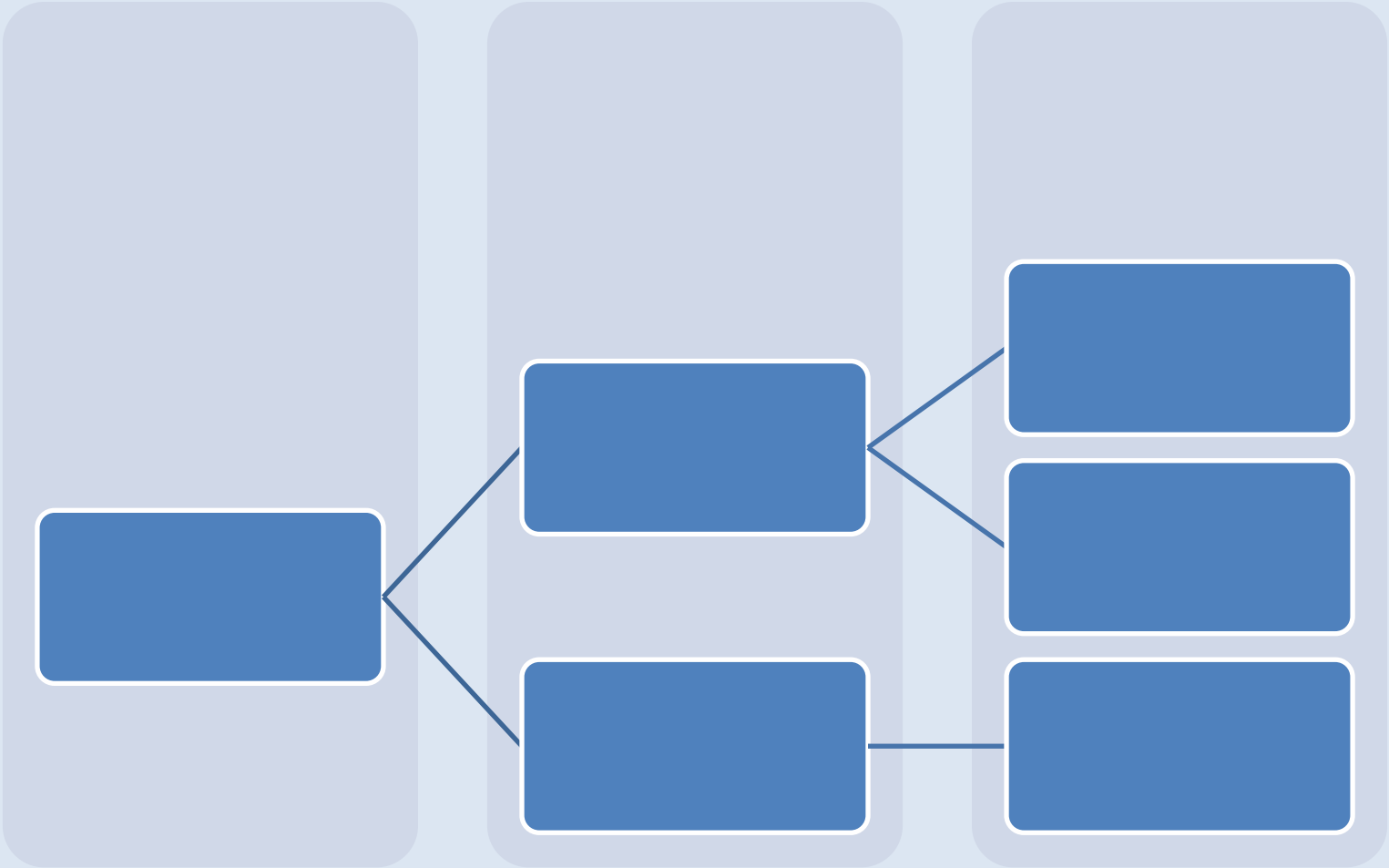
ADAGRAD often improves SGD convergence on power-law data **by orders of magnitude**.

Outline

- Supervised Learning
  - Linear Regression
    - R-squared
  - Logistic Regression
  - SVMs
- Gradient-based optimization
- Decision Trees and Random Forests

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$





Choosing best features in your dataset

At each node, we choose the feature f which **maximizes the information gain**.

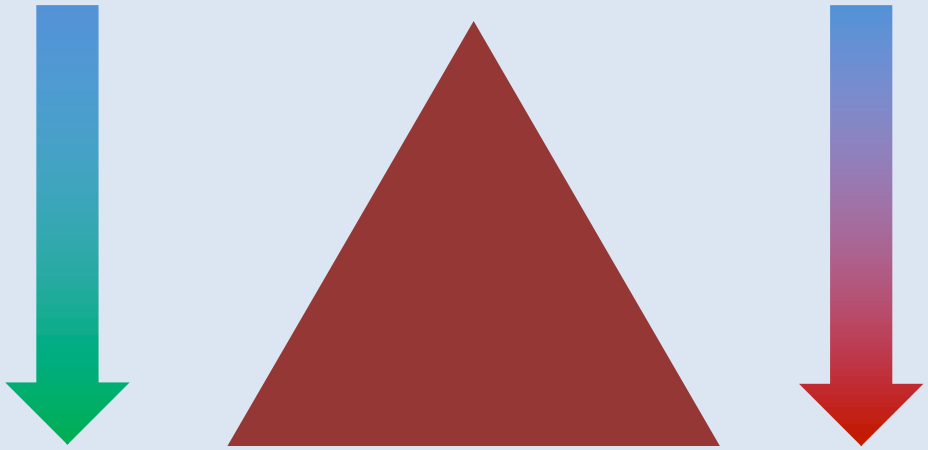
This tends to produce mixtures of classes at each node that **are more and more "pure" (less noise more signal)** as you go down the tree.

If a node has examples all of one class c, we make it a leaf and output "c".

Otherwise, when we hit the depth limit, we output **the most popular class** at that node.

Decision Tree Models

- As tree depth increases, bias decreases and variance generally increases. Why? (Hint: think about k-NN)



Bias decreases with tree depth

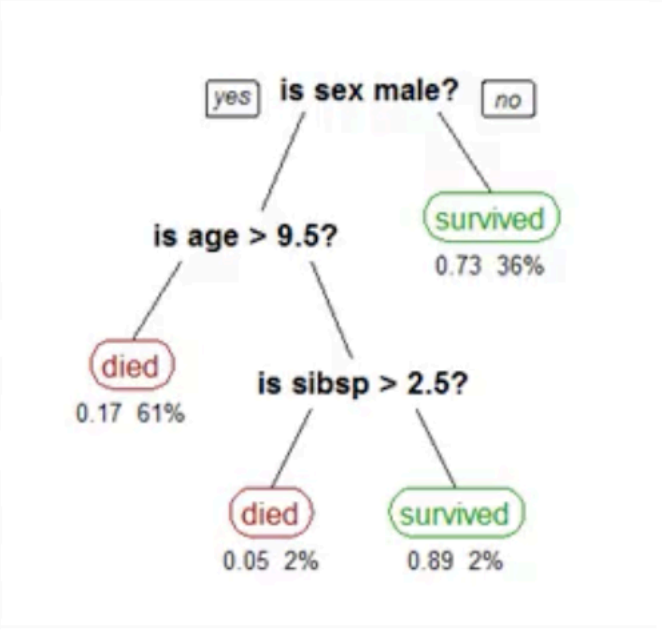
Variance increases with tree depth

Decision Trees

- Graph like structure where nodes represent decisions to split on, and leaves represent the outcome
- The feature to split on is selected based on Information Gain or Entropy
- The basic principle is to reduce impurity or to increase information gain as much as possible
- Results in lower variance in the branches after splitting, meaning more points from the same class leads to lower entropy values (less impurity)
- The first feature to split on is the most important one that results in largest decrease in impurity

Decision Trees

- Entropy = $-\sum_i (p_i) \log_2 (p_i)$
- Information Gain = Entropy Parent – Weighted Average of Entropy Children
- Goal: Maximize Information Gain



Ensemble Methods

Are like **Crowdsourced machine learning algorithms**:

- Take a collection of simple or *weak* learners
- Combine their results to make a single, better learner

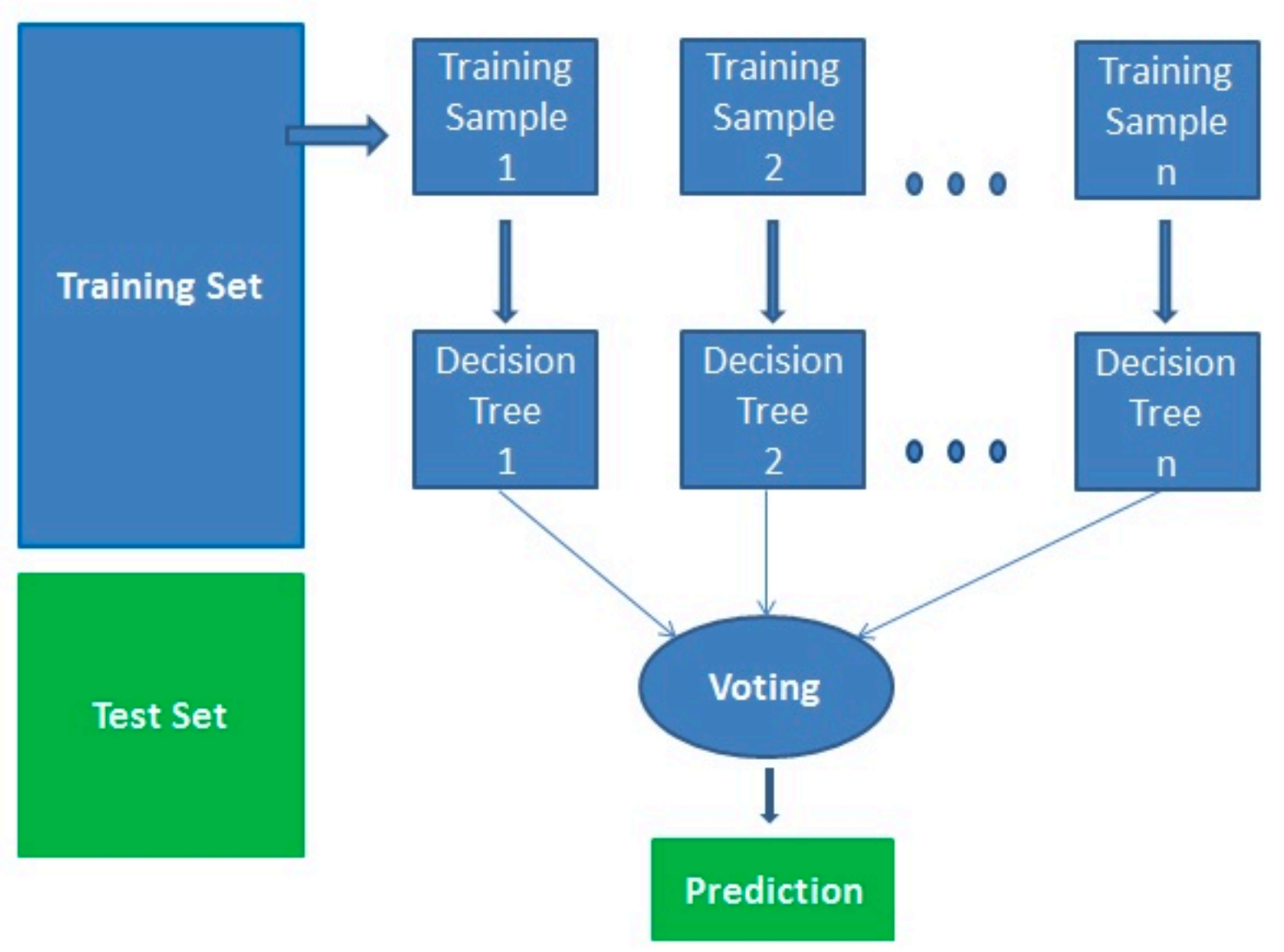
Types:

- **Bagging:** train learners in parallel on different samples of the data, then combine by voting (discrete output) or by averaging (continuous output).
- **Stacking:** combine model outputs using a second-stage learner like linear regression.
- **Boosting:** train learners on the filtered output of other learners. (XGBoost)

Random Forests

1. Select random samples (batches) from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.

N-fold cross validation



Advantages:

- Random forests is considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
- The algorithm can be used in both classification and regression problems.
- Random forests can also handle missing values.
  - There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.
- You can get the relative feature importance, which helps in selecting the most contributing features for the classifier.

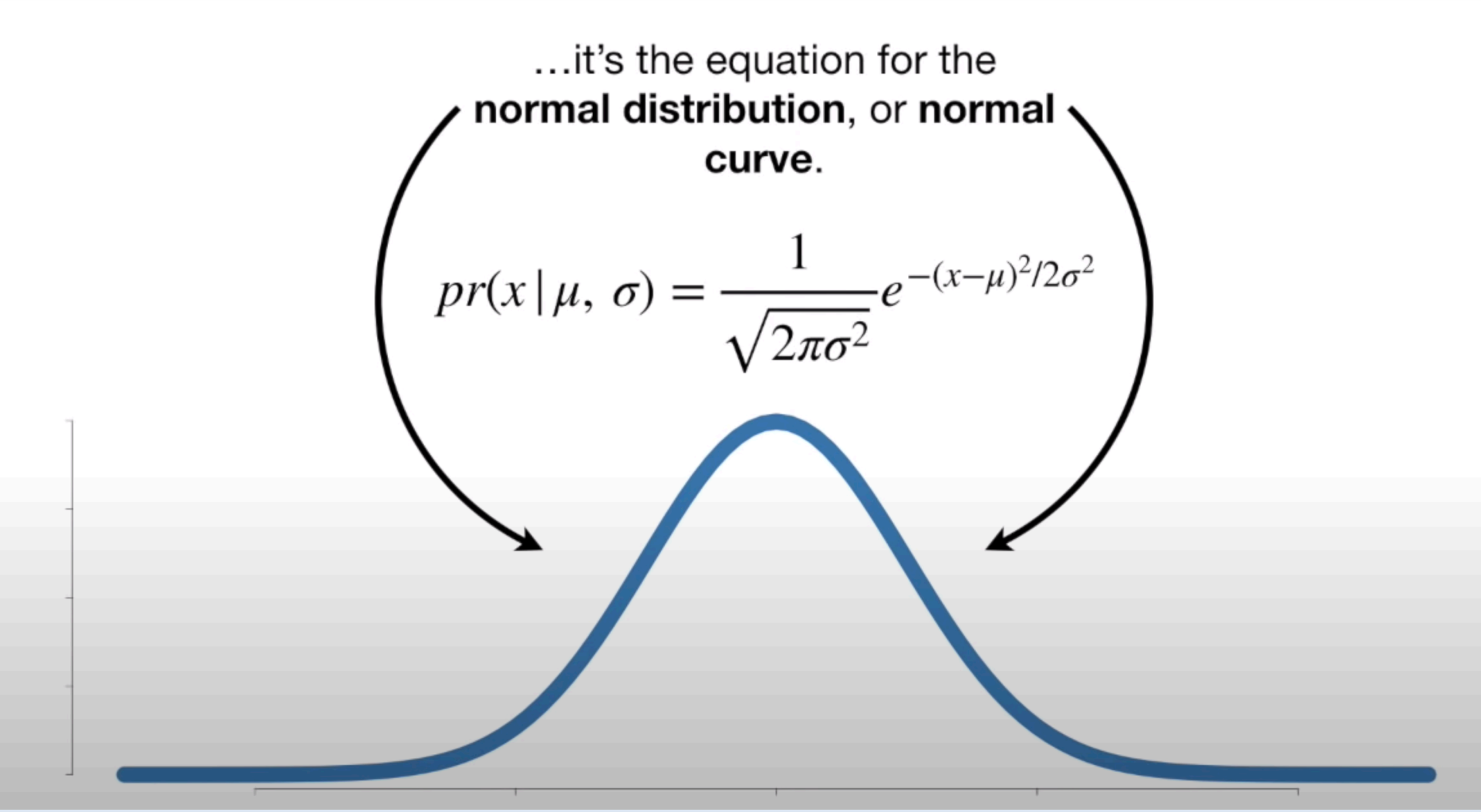
Disadvantages:

- Random forests is slow in generating predictions because it has multiple decision trees.
- Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it.
- This whole process is time-consuming.
- The model is difficult to interpret compared to a decision tree, where you can easily make a decision by following the path in the tree.
- ** if you want explainability you need to record the vote from each subtree, so you can go back and justify the reason why you classified in a certain way.*

Finding important features

- Random forest uses **gini** importance or mean decrease in impurity (MDI) to calculate the importance of each feature.
- Gini importance is also known as the total decrease in node impurity.
  - This is how much the model fit or accuracy decreases when you drop a variable.
- The larger the decrease, the more significant the variable is.
  - Here, the mean decrease is a significant parameter for variable selection.
- The Gini index can describe the overall explanatory power of the variables.
- Random forests also offers a good feature selection indicator.
  - Scikit-learn provides an extra variable with the model, which shows the relative importance or contribution of each feature in the prediction.
  - It automatically computes the relevance score of each feature in the training phase.
- Then it scales the relevance down so that the sum of all scores is 1.
- This score will help you choose the most important features and drop the least important ones for model building. *(PCA may give you something somewhat similar)*

Probability Distributions

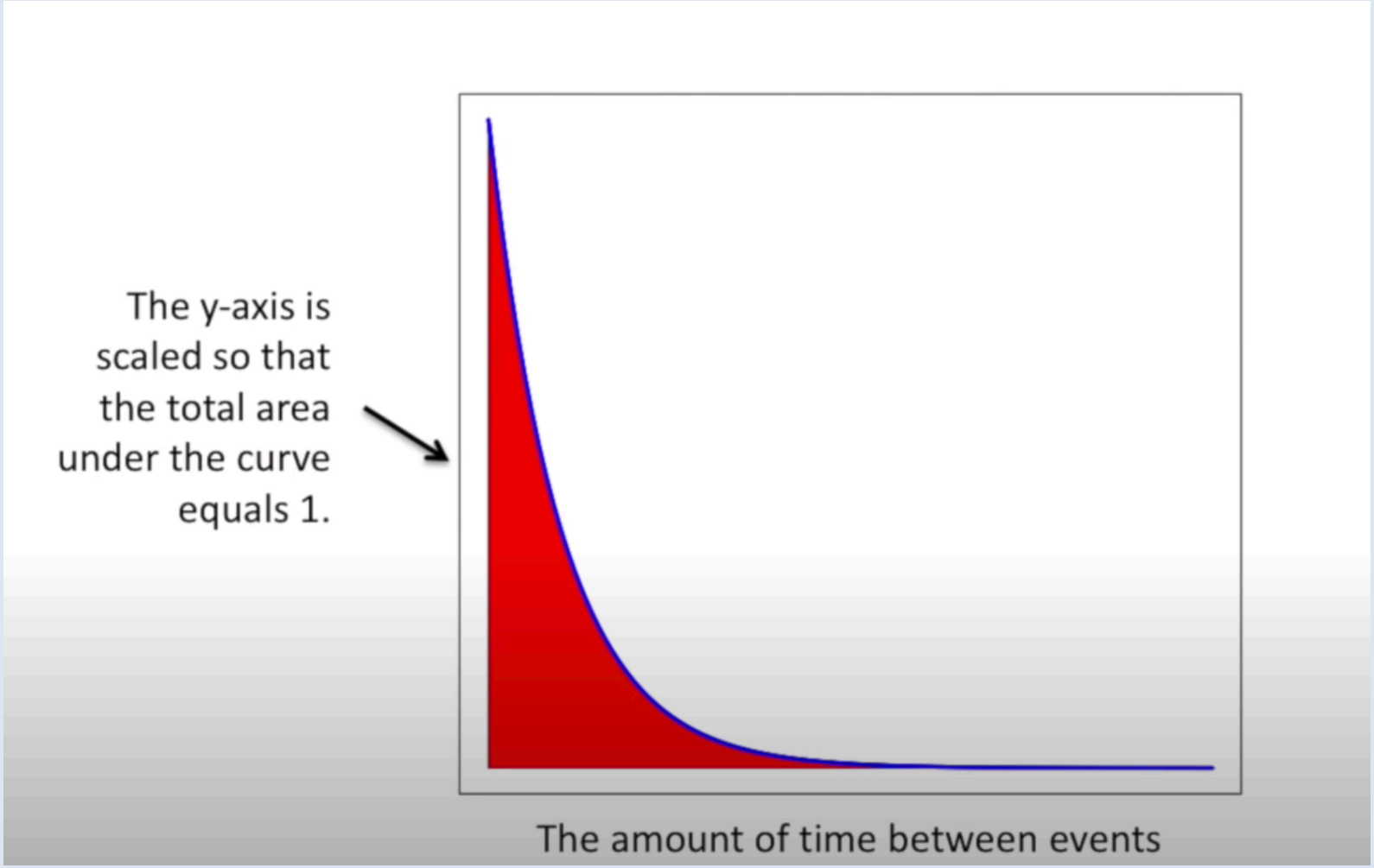


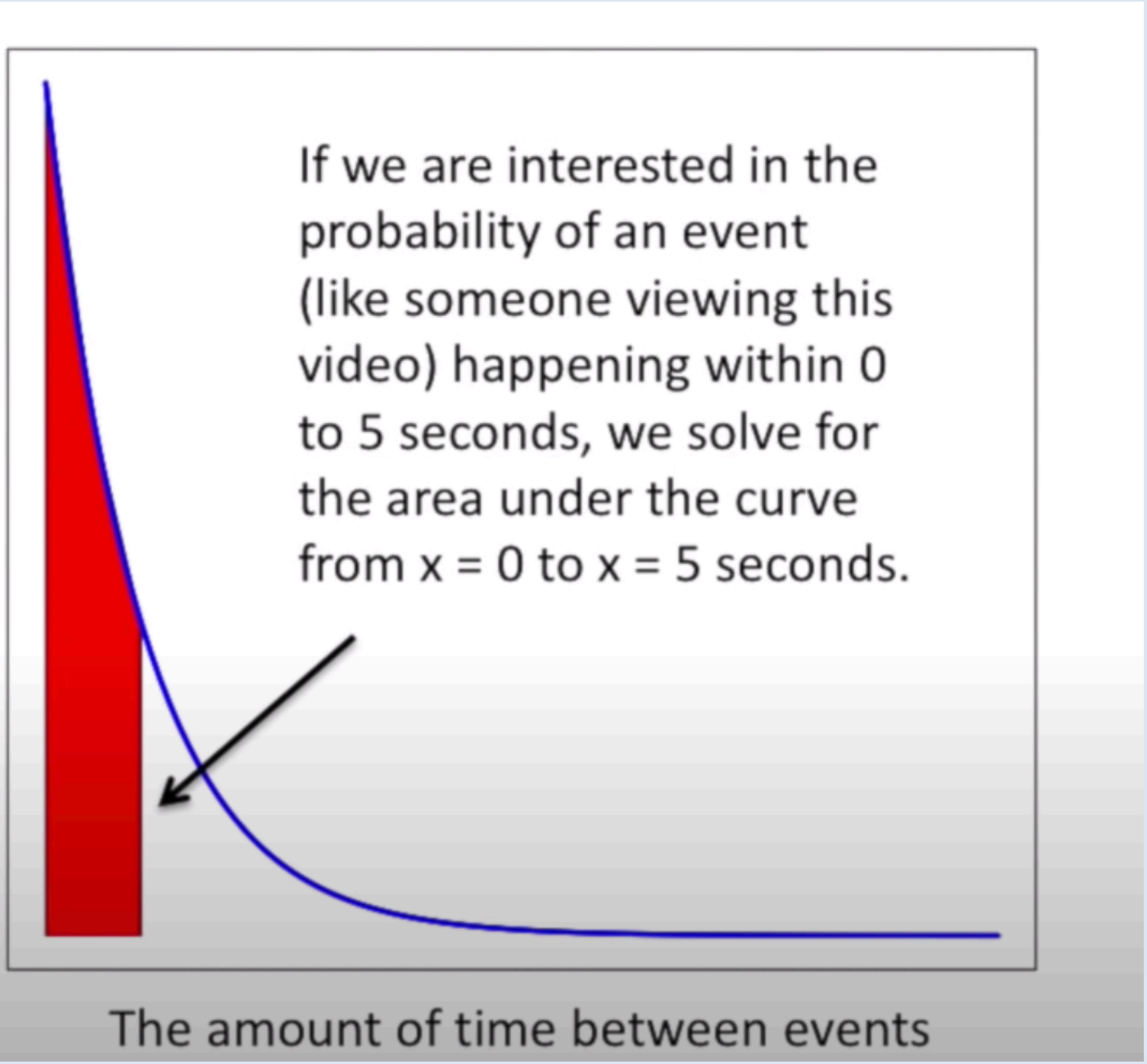
…to find the optimal values for $\boldsymbol{\mu}$ (the mean) and $\boldsymbol{\sigma}$ (the standard deviation) given some data, $\boldsymbol{x}$

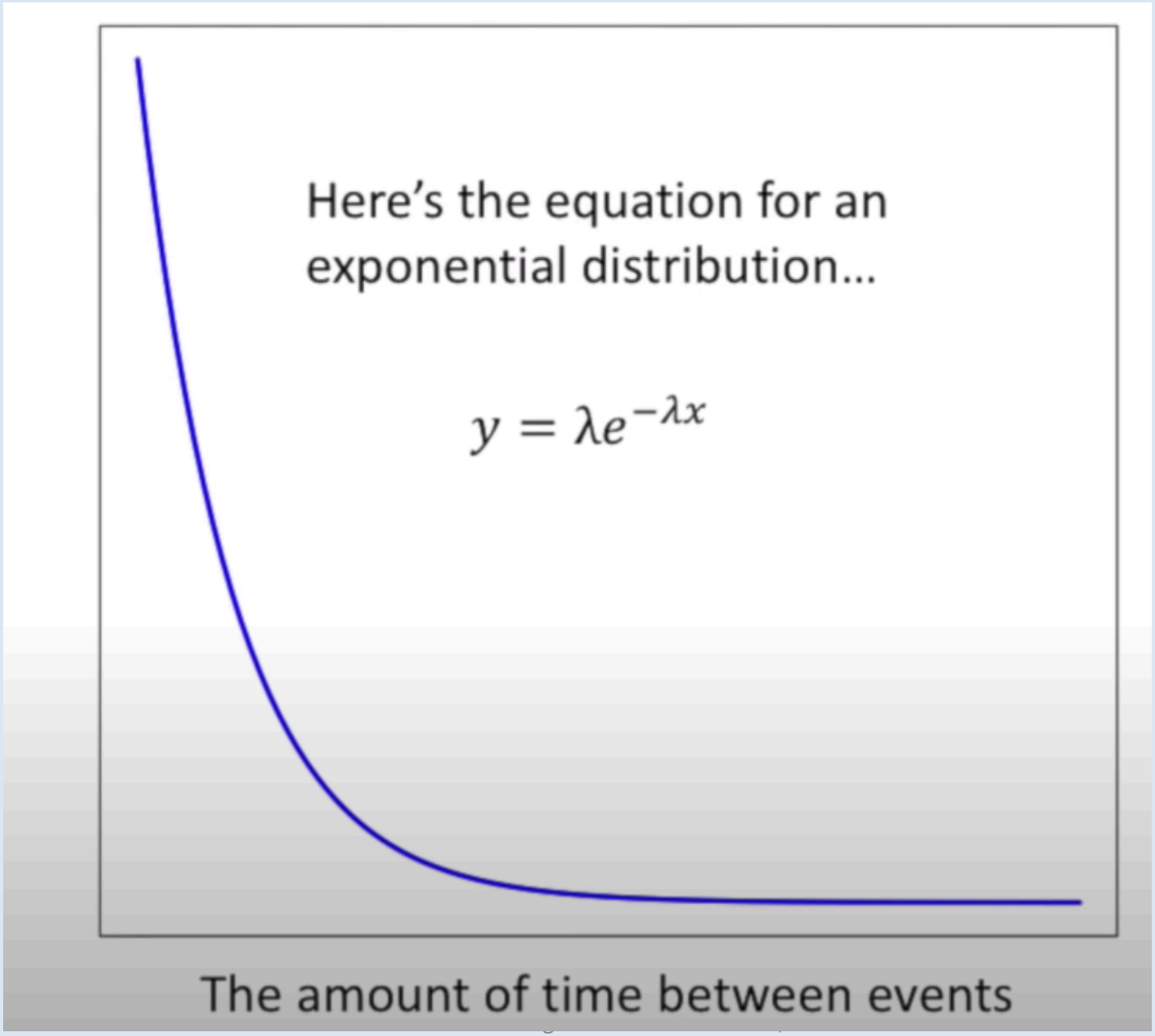
$$pr(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

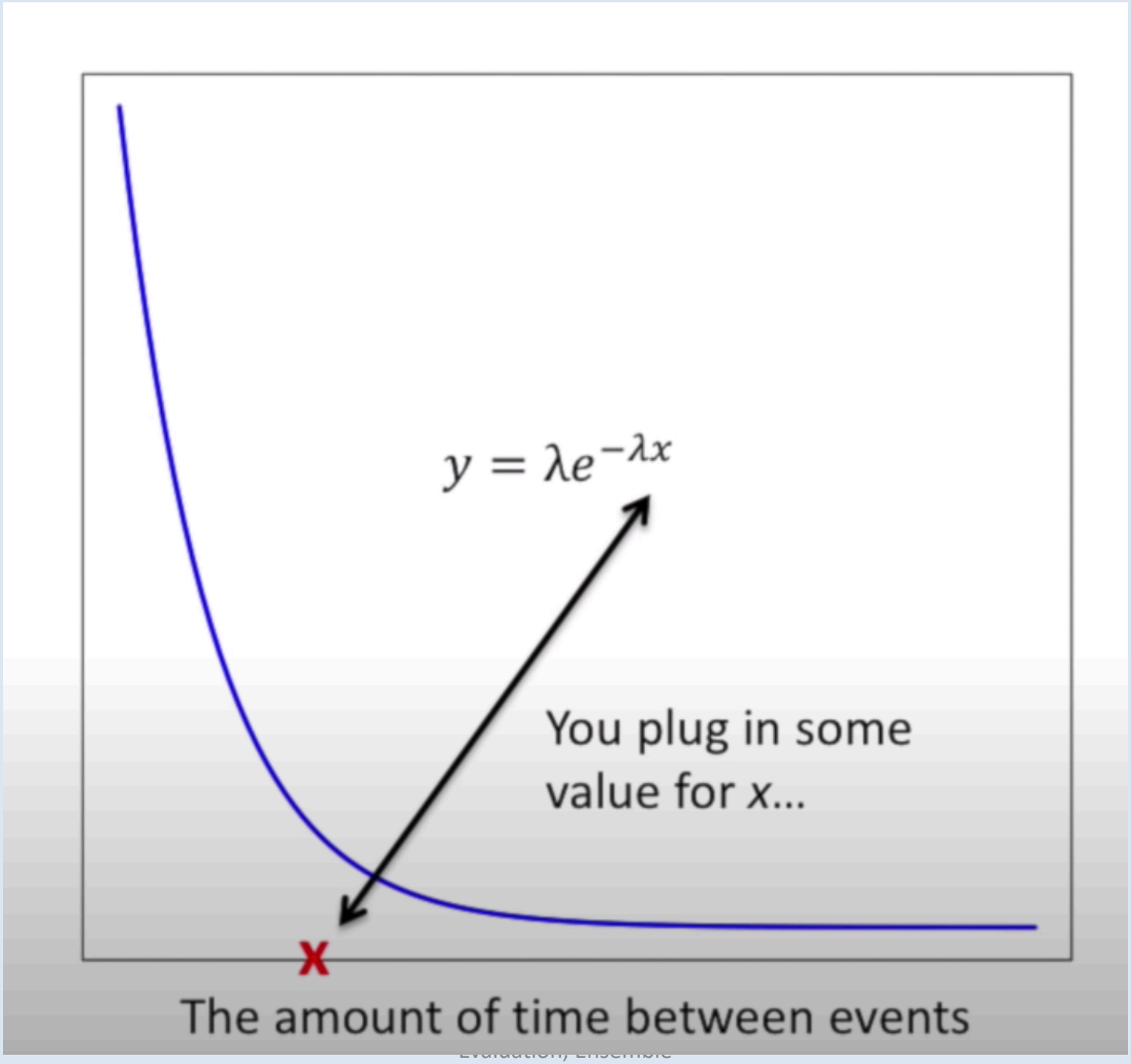
$$L(\mu, \sigma \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

Exponential Distribution



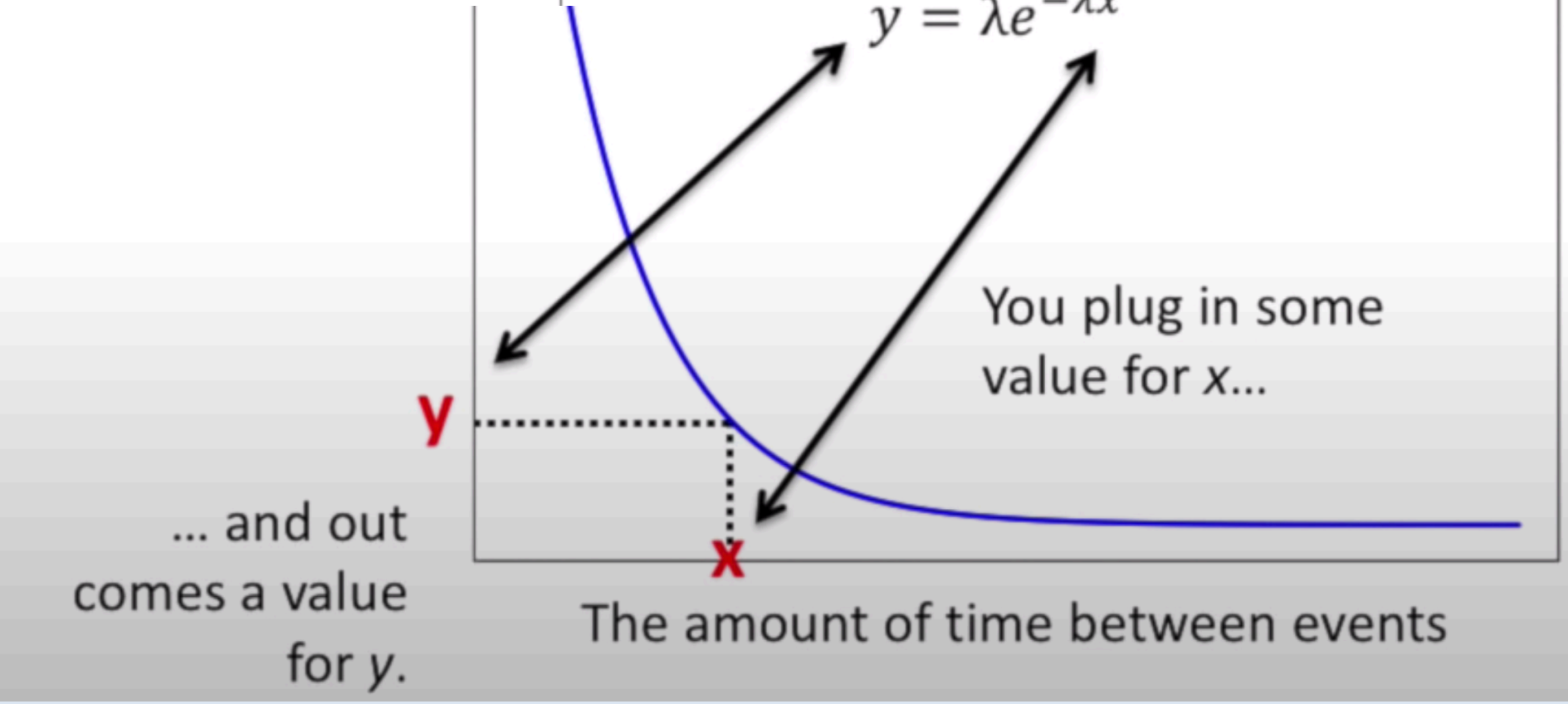

If we are interested in the probability of an event (like someone viewing this video) happening within 0 to 5 seconds, we solve for the area under the curve from x = 0 to x = 5 seconds.
The amount of time between events

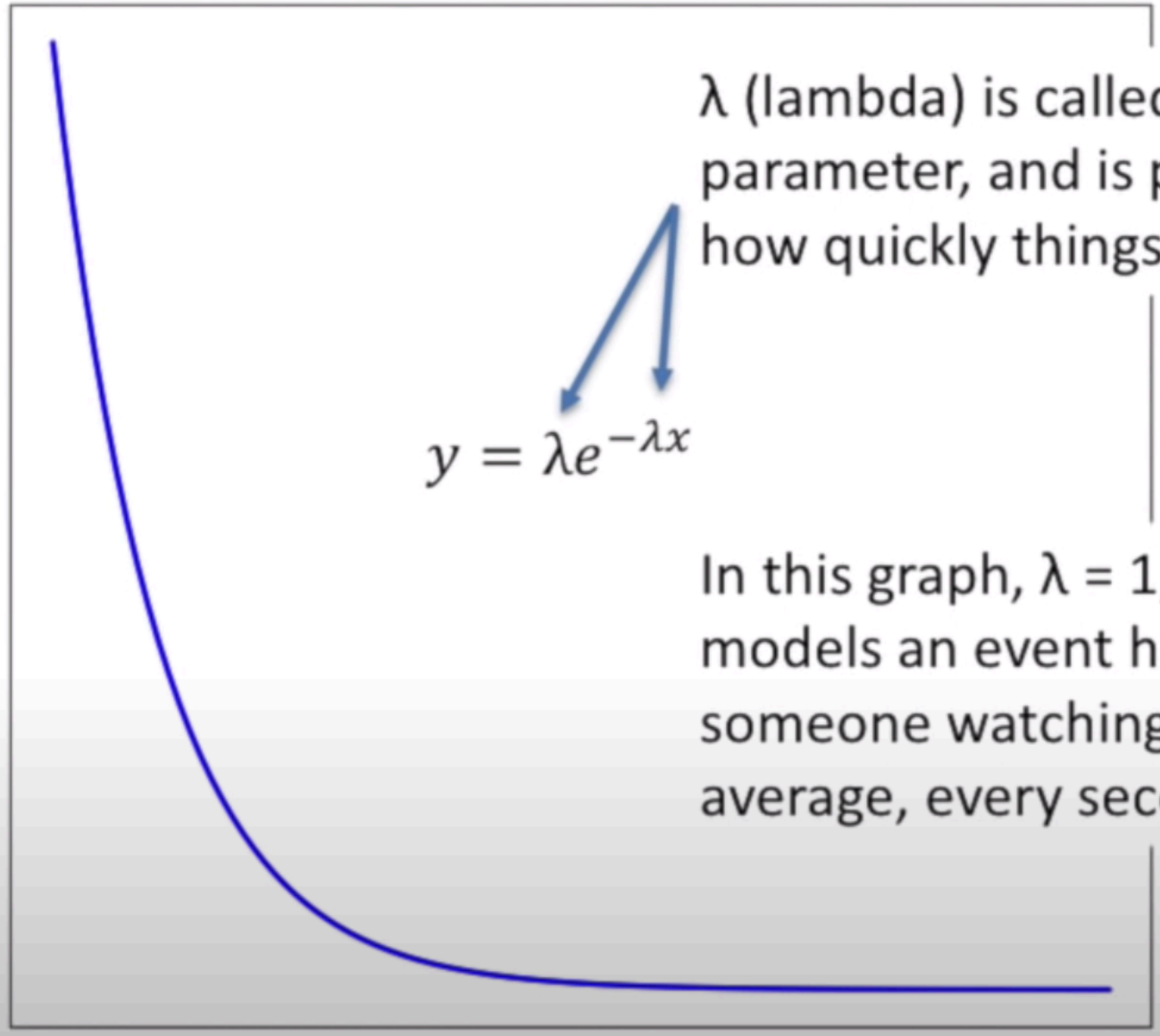

Here's the equation for an exponential distribution…
$y = \lambda e^{-\lambda x}$
The amount of time between events


$y = \lambda e^{-\lambda x}$
You plug in some value for x…
X
The amount of time between events

λ (lambda) is called the "rate" parameter, and is proportional to how quickly things happen.

$y = \lambda e^{-\lambda x}$



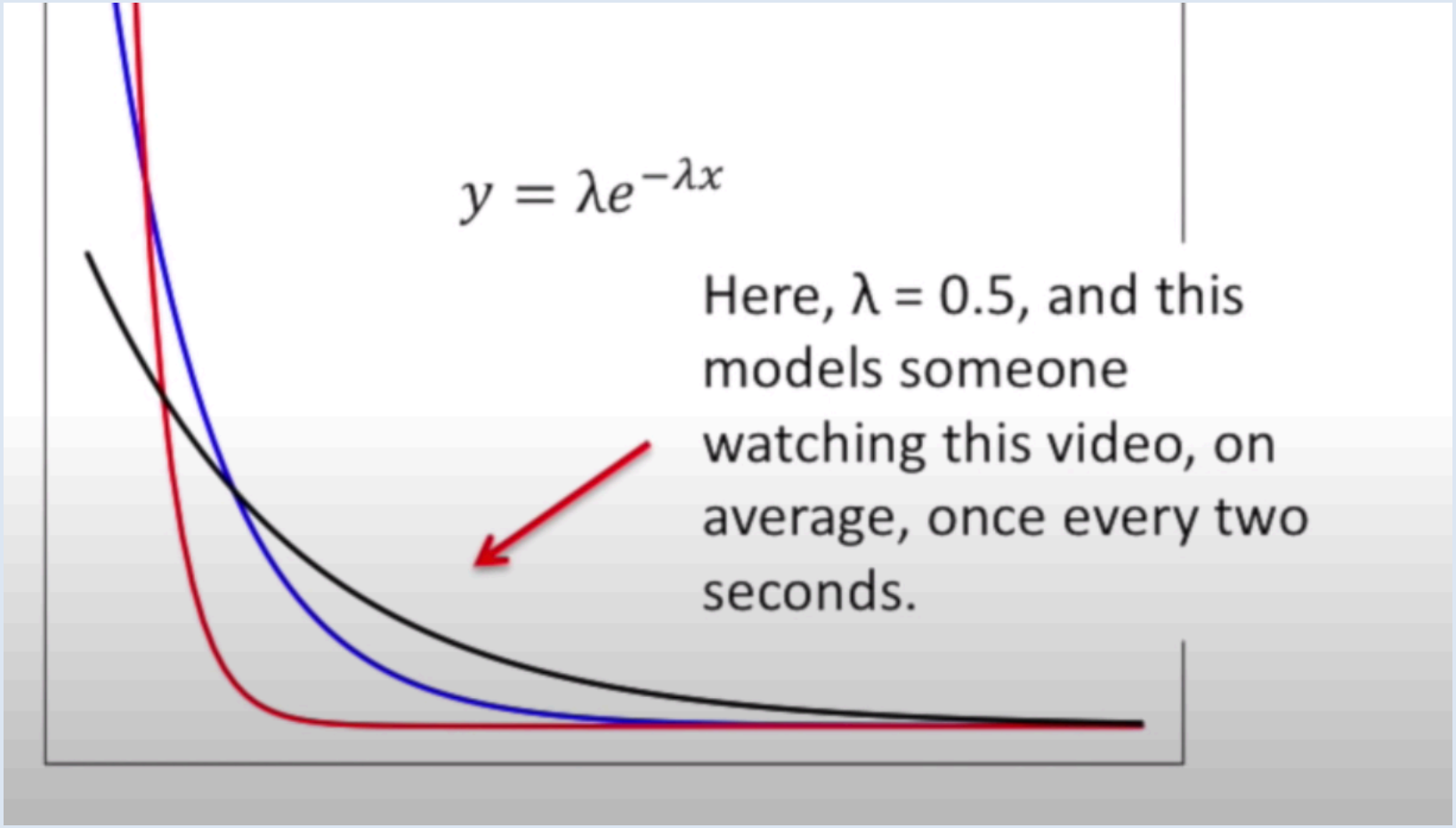


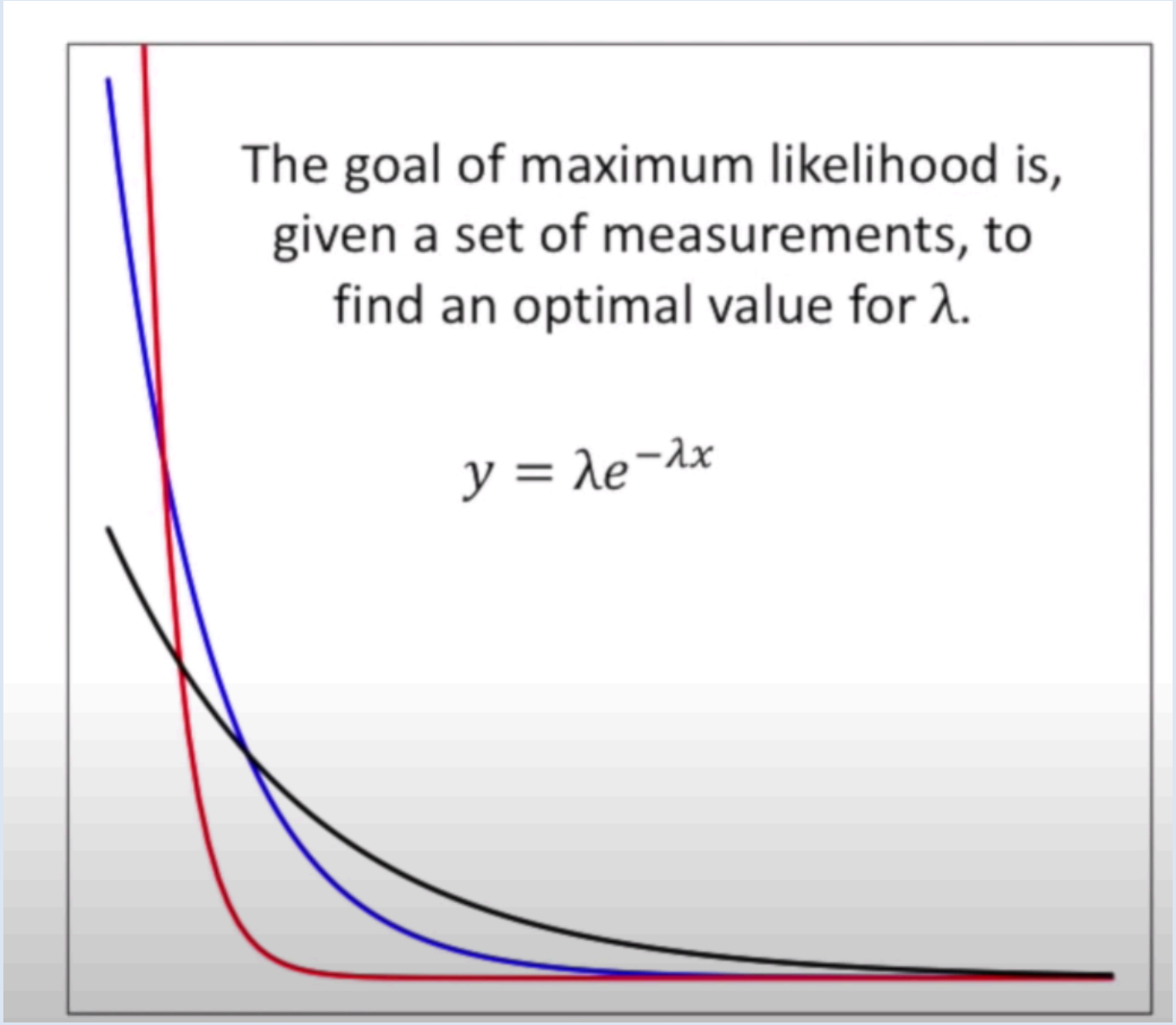
λ (lambda) is called the "rate" parameter, and is proportional to how quickly things happen.

$$y = \lambda e^{-\lambda x}$$

In this graph, λ = 1, and this models an event happening (like someone watching this video), on average, every second.

The amount of time between events


$y = \lambda e^{-\lambda x}$
Here, λ = 0.5, and this models someone watching this video, on average, once every two seconds.


The goal of maximum likelihood is, given a set of measurements, to find an optimal value for λ.
$y = \lambda e^{-\lambda x}$

So assume I collected a lot of data about how much time passed between views of this video.

x_1 = the amount of time that passed between the 1st and 2nd views.

x_2 = the amount of time that passed between the 2nd and 3rd views.

x_3 = the amount of time that passed between the 3rd and 4th views.

etc. etc. etc.

Here are the 'n' measurements…

$x_1, x_2, x_3, \ldots, x_n$

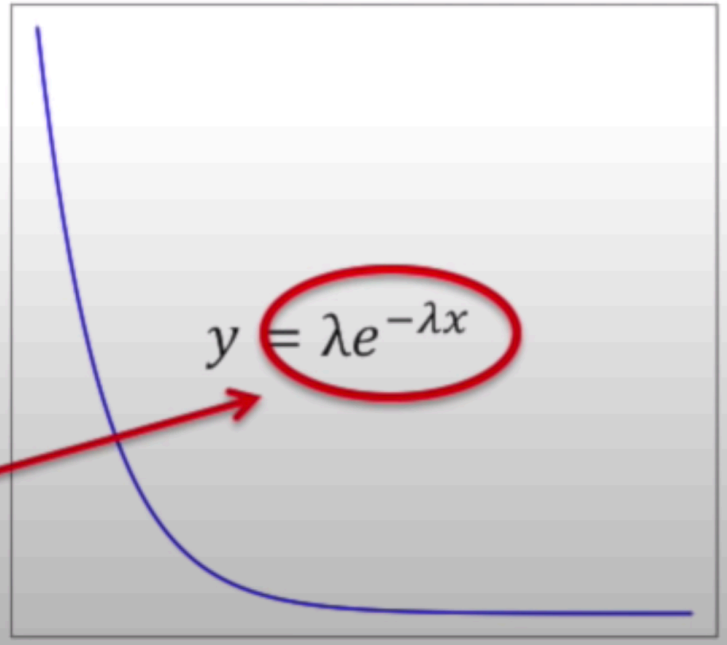
For now, let's assume that we already have a good value for λ.

What is the likelihood of λ given our first measurement, x_1?

$L(\lambda \mid x_1) = \lambda e^{-\lambda x_1}$

This equation is just…

…the equation for the curve with x_1 plugged into it.

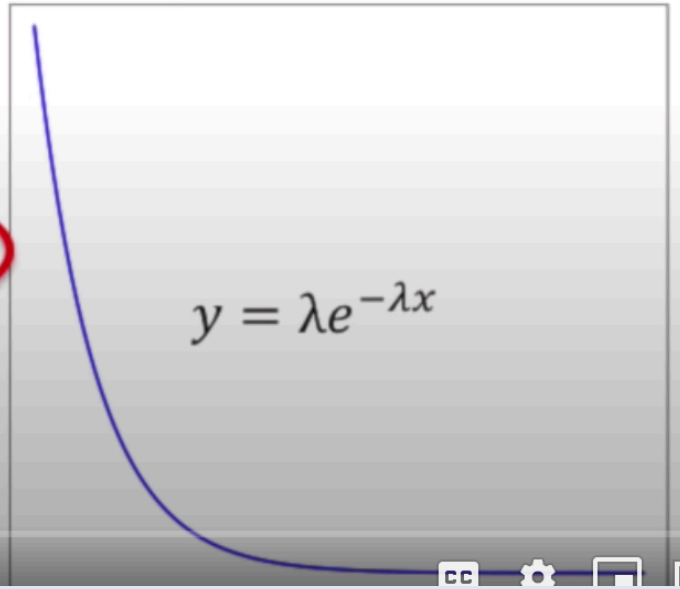


What is the likelihood of λ given our first measurement, x_1?

$$L(\lambda \mid x_1) = \lambda e^{-\lambda x_1}$$

$$L(\lambda \mid x_2) = \lambda e^{-\lambda x_2}$$

Similarly, the likelihood of λ given the second measurement, x_2 is just the equation for the curve with x_2 plugged into it.



3:48 / 9:38

Because we are interested in x_1 and x_2, we multiply the two likelihood functions together.

$$L(\lambda \mid x_1) = \lambda e^{-\lambda x_1}$$

$$L(\lambda \mid x_2) = \lambda e^{-\lambda x_2}$$

$$L(\lambda \mid x_1 \text{ and } x_2) = L(\lambda \mid x_1) L(\lambda \mid x_2)$$

$$L(\lambda \mid x_1) = \lambda e^{-\lambda x_1} \qquad L(\lambda \mid x_2) = \lambda e^{-\lambda x_2}$$

$$L(\lambda \mid x_1 \text{ and } x_2) = L(\lambda \mid x_1) L(\lambda \mid x_2)$$

$$= \lambda e^{-\lambda x_1} \lambda e^{-\lambda x_2}$$

$$= \lambda^2 \left[e^{-\lambda x_1} e^{-\lambda x_2} \right]$$

$$= \lambda^2 \left[e^{-\lambda (x_1 + x_2)} \right]$$

Thus, this equation is the likelihood of λ given x_1 and x_2.

What is the likelihood of λ given all of the data, $x_1, x_2, \ldots, x_n$?

Here's the likelihood function that includes all of the data we have collected.

$$L(\lambda \mid x_1, x_2, \ldots, x_n)$$

It is equal to the product of all the individual likelihoods.

$$L(\lambda \mid x_1, x_2, \ldots, x_n) = L(\lambda \mid x_1)L(\lambda \mid x_2) \ldots L(\lambda \mid x_n)$$

$$L(\lambda \mid x_1, x_2, \ldots, x_n) = L(\lambda \mid x_1) L(\lambda \mid x_2) \ldots L(\lambda \mid x_n)$$

$$= \lambda e^{-\lambda x_1} \lambda e^{-\lambda x_2} \ldots \lambda e^{-\lambda x_n}$$

Here I've just plugged in all the individual likelihood equations…

$$L(\lambda \mid x_1, x_2, \ldots, x_n) = L(\lambda \mid x_1) L(\lambda \mid x_2) \ldots L(\lambda \mid x_n)$$

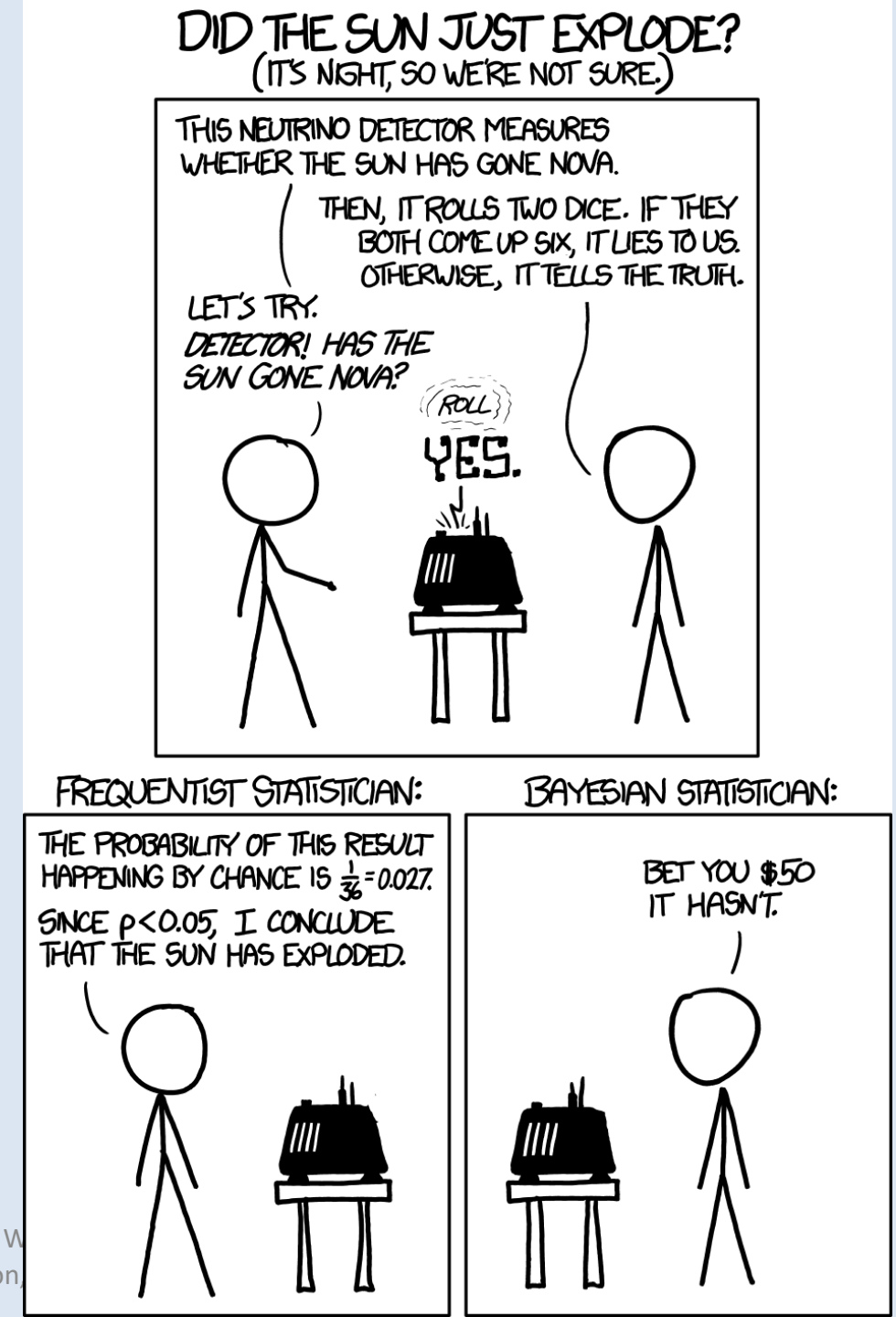
$$= \lambda e^{-\lambda x_1} \lambda e^{-\lambda x_2} \ldots \lambda e^{-\lambda x_n}$$

$$= \lambda^n \left[e^{-\lambda x_1} e^{-\lambda x_2} \ldots e^{-\lambda x_n} \right]$$

$$= \lambda^n \left[e^{-\lambda (x_1 + x_2 + \ldots + x_n)} \right]$$

Thus, this equation is the likelihood of λ given all of the data, $x_1, x_2, \ldots x_n$.

A Gentle Introduction to Maximum Likelihood Estimation



Setting Up Our Problem

To approach MLE today, let's come from the Bayesian angle, and use Bayes Theorem to frame our question as such:

```
P(β|y) = P(y|β) x P(β) / P(y)
```

We can effectively ignore the prior and the evidence because — given the Wiki definition of a uniform prior distribution — all coefficient values are equally likely. And probability of all data values (assume continuous) are equally likely, and basically zero.

Or, in English:

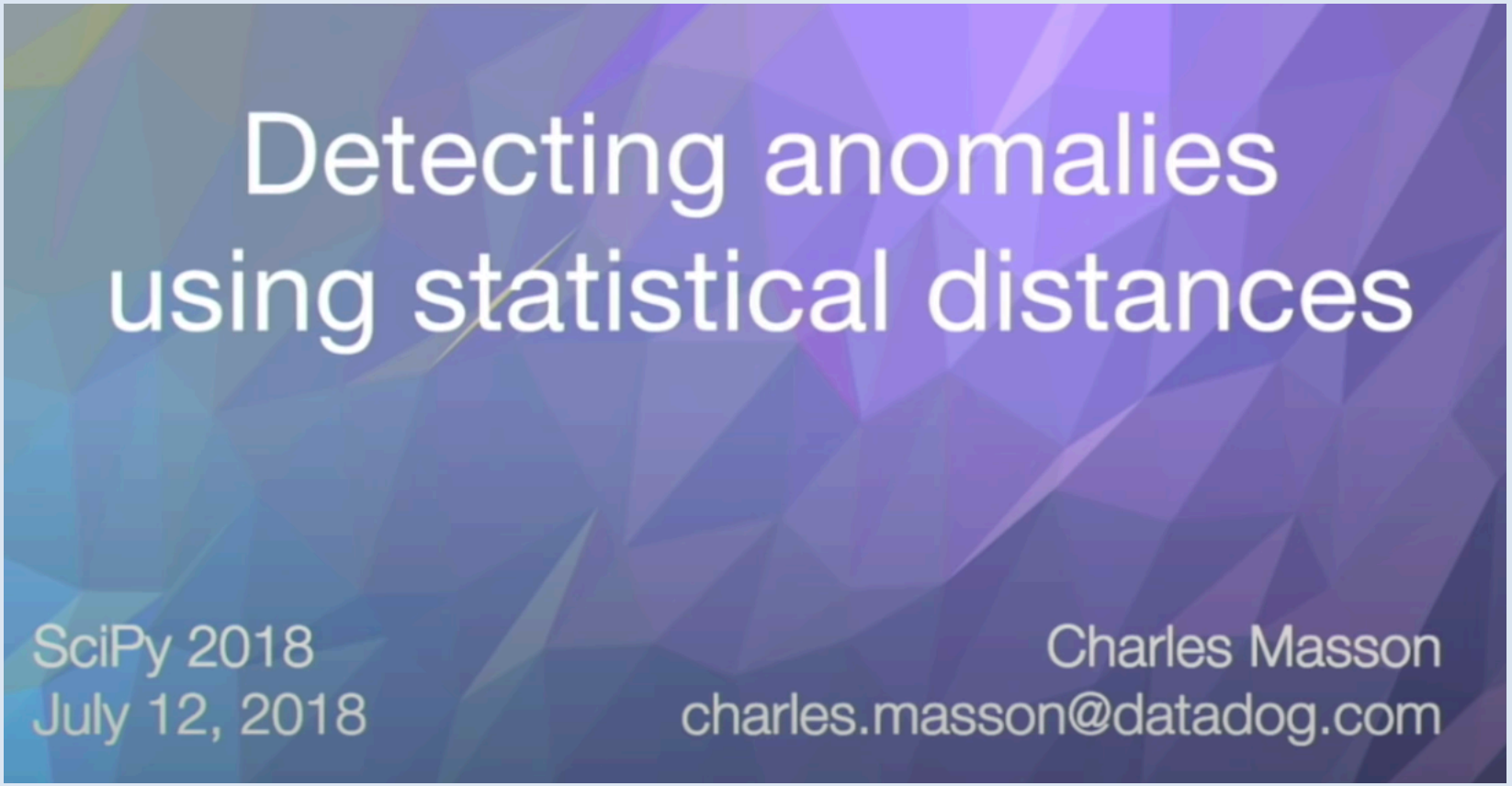
```
posterior = likelihood x prior / evidence
```

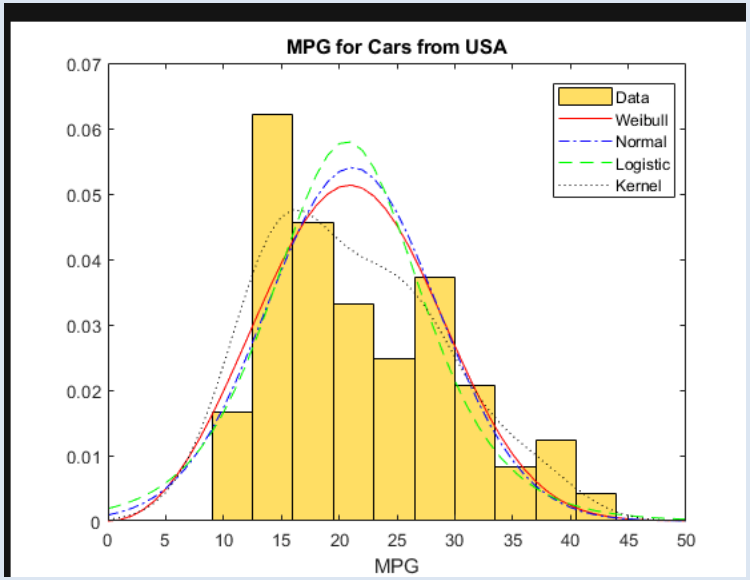
Probability density functions[1]

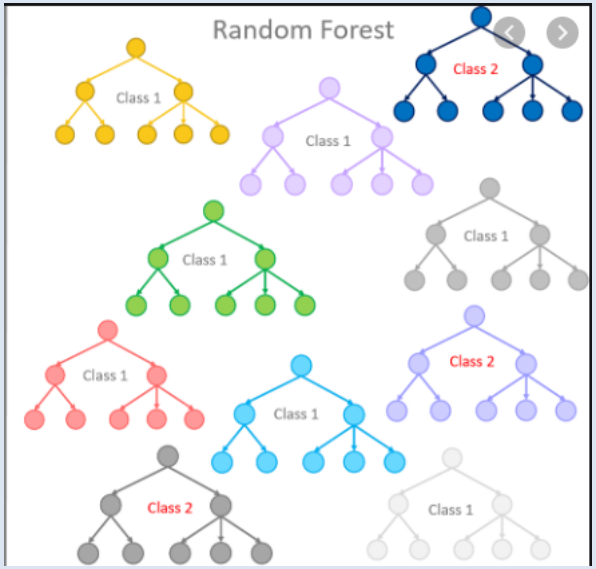
- Probability density function's value at some specific point does not give you probability;
- it is a measure of how *dense* the distribution is around that value.
- For continuous random variables, the probability at a given point is equal to zero.
- Instead of $p(X = x)$, we calculate probabilities between 2 points $p(x1 < X < x2)$ and it is equal to the area below that probability density function.
- Probability density function's value can very well be above 1. It can even approach to infinity.

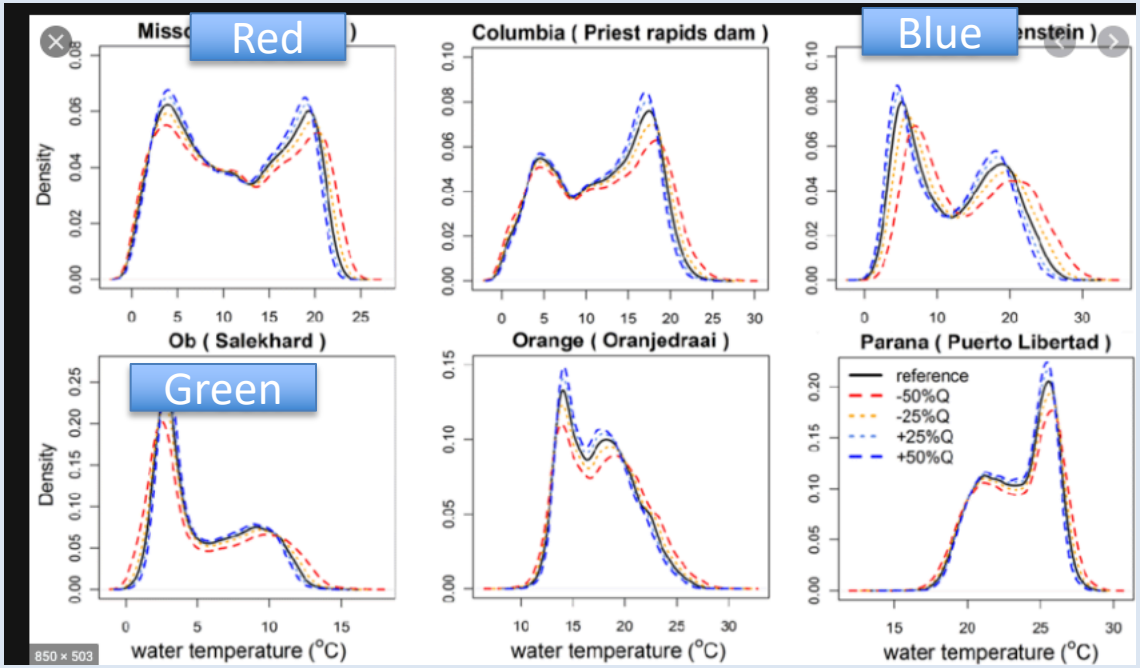
The Notion of Distance and Similarity in Machine Learning

https://www.youtube.com/watch?v=U7xdiGc7IRU

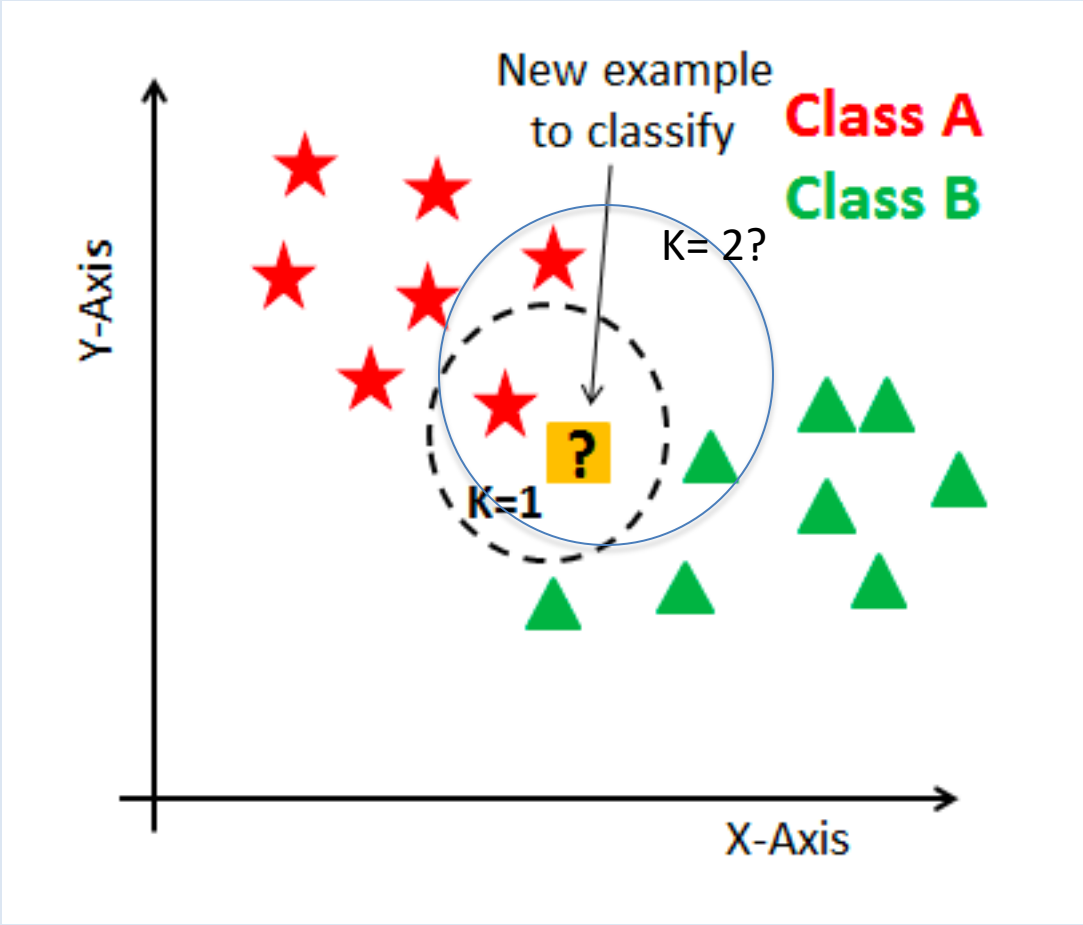



MPG for Cars from USA
Data
Weibull
Normal
Logistic
Kernel
MPG


Random Forest
Class 1
Class 2


Red
Blue
Green
Columbia (Priest rapids dam)
Ob (Salekhard)
Orange (Oranjedraai)
Parana (Puerto Libertad)
reference
-50%Q
-25%Q
+25%Q
+50%Q
Density
water temperature (°C)
850 × 503

K Nearest Neighbors vs Naïve Bayes

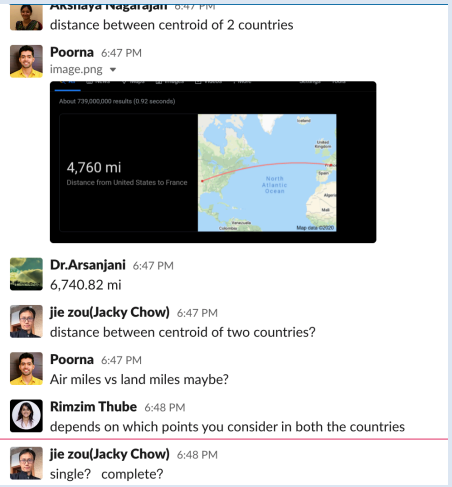


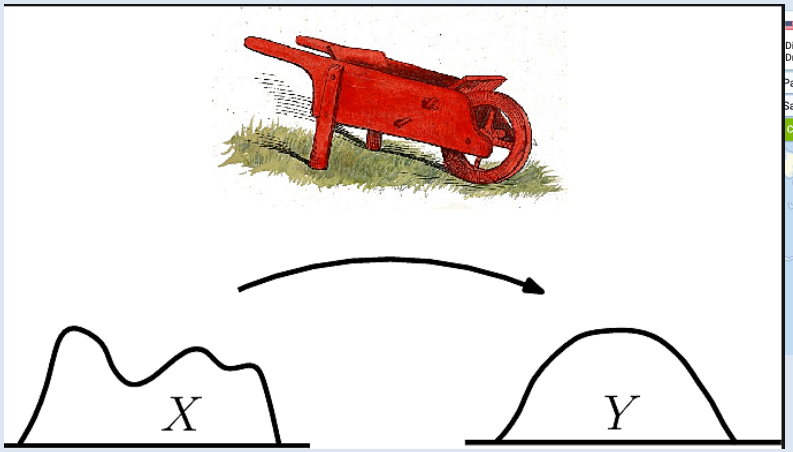
Source: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

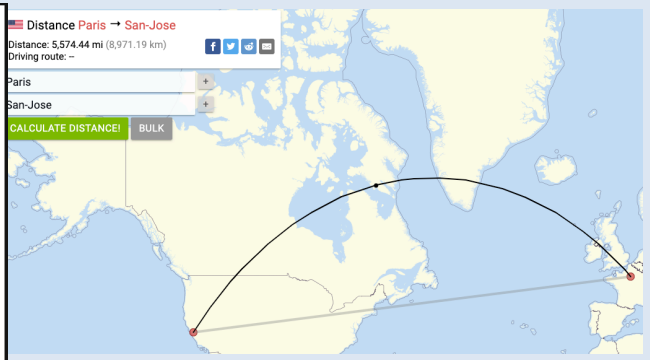
Neighbors and Distance

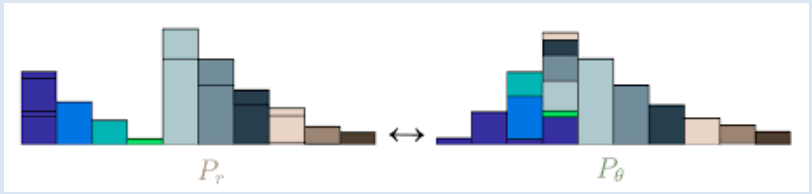
- Suppose P1 is the point, for which label needs to predict.
  - First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors.
- Each object votes for their class and the class with the most votes is taken as the prediction.
- For finding closest similar points, you find the distance between points using:
  - Euclidean distance,
  - Hamming distance,
  - Manhattan distance
  - Minkowski distance.

Question: distance bet two prob dist





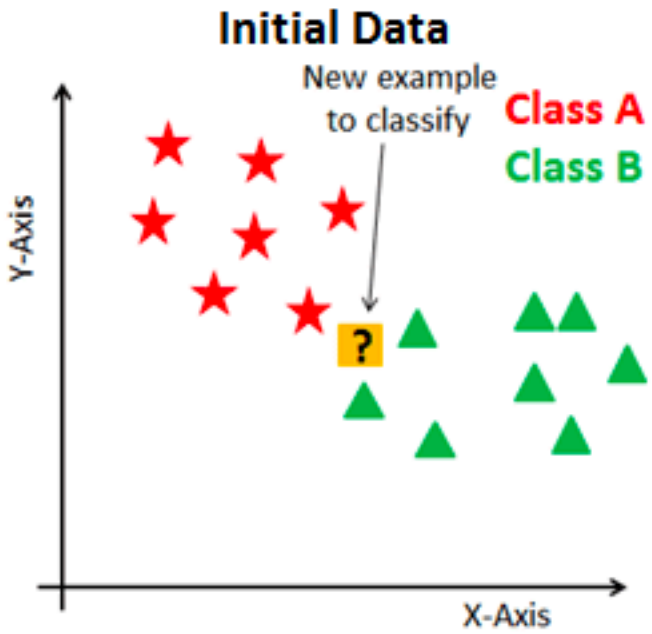


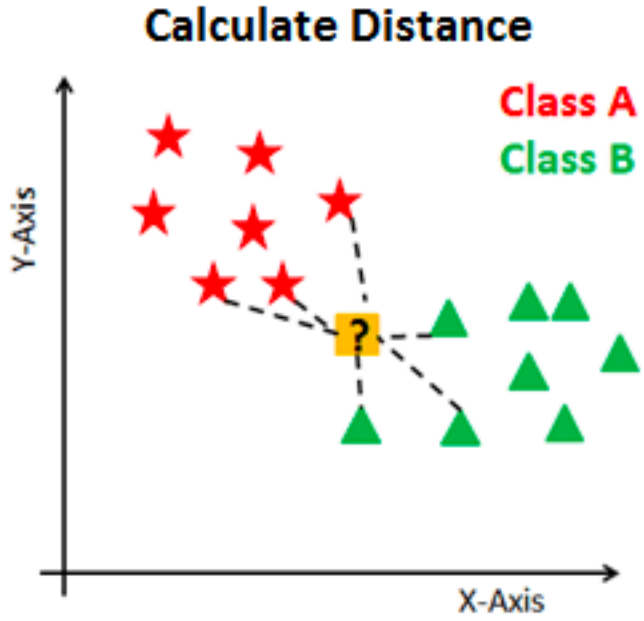


US

Cross entropy
KL Div
Wasserstein Distance

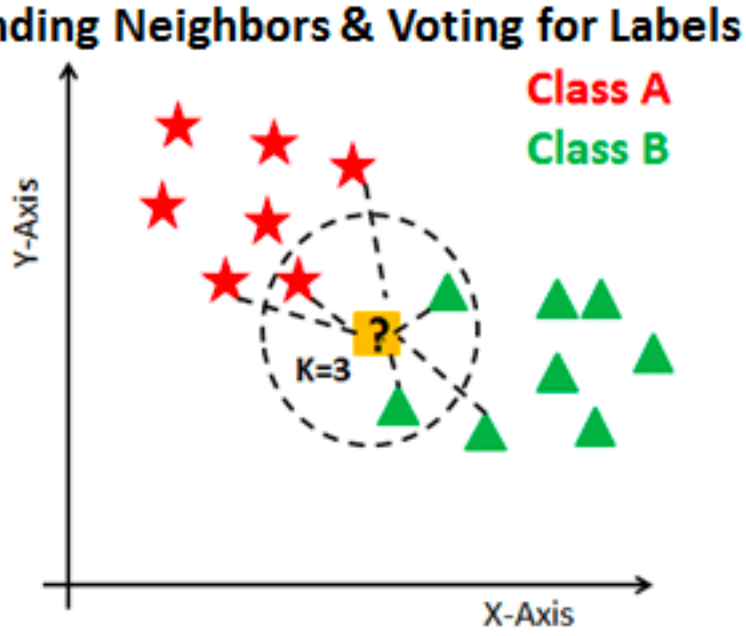
France





KNN steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels



Source: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

Eager vs Lazy Learning

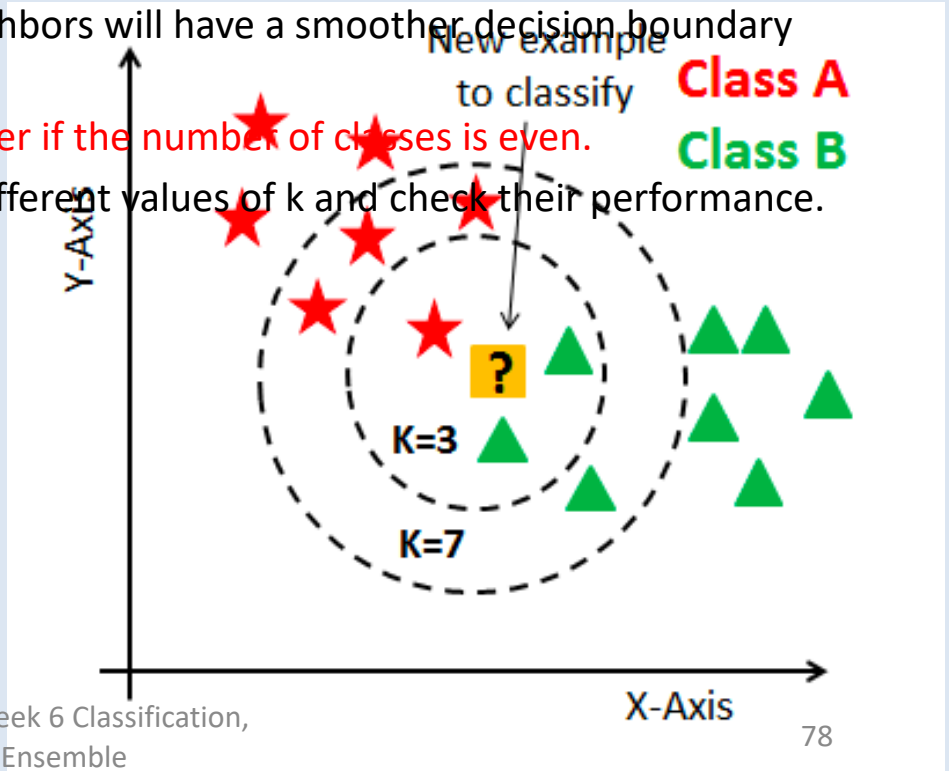
- …means there is no need for learning or training of the model and all of the data points used at the time of prediction.
- Lazy learners wait until the last minute before classifying any data point.
- Lazy learner stores merely the training dataset and waits until classification needs to perform.
- Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples.
- Unlike eager learning methods, lazy learners do less work in the training phase and more work in the testing phase to make a classification.
- Lazy learners are also known as instance-based learners because lazy learners store the training points or instances, and all learning is based on instances.

Curse of Dimensionality

- KNN performs better with a lower number of features than a large number of features.
- You can say that when the number of features increases than it requires more data.
- Increase in dimension also leads to the problem of overfitting.
- To avoid overfitting, the needed data will need to grow exponentially as you increase the number of dimensions.
- This problem of higher dimension is known as the Curse of Dimensionality.
- To deal with the problem of the curse of dimensionality, you need to perform principal component analysis before applying any machine learning algorithm, or you can also use feature selection approach.
- Research has shown that in large dimension Euclidean distance is not useful anymore.
- Therefore, you can prefer other measures such as cosine similarity, which get decidedly less affected by high dimension.

number of neighbors in KNN?

- How to choose the optimal number of neighbors?
- And what are its effects on the classifier?
- The number of neighbors(K) in KNN is a hyperparameter that you need choose at the time of model building. You can think of K as a controlling variable for the prediction model.
- Research has shown that no optimal number of neighbors suits all kind of data sets.
- Each dataset has it's own requirements.
- In the case of a small number of neighbors, the noise will have a higher influence on the result, and a large number of neighbors make it computationally expensive.
- Research has also shown that a small amount of neighbors are most flexible fit which will have low bias but high variance and a large number of neighbors will have a smoother decision boundary which means lower variance but higher bias.
- Generally, Data scientists choose as an odd number if the number of classes is even.
- You can also check by generating the model on different values of k and check their performance.
- You can also try Elbow method here.



Example KNN-ML.ipny

- K-Nearest Neighbor algorithm;
- it's working, eager and lazy learner,
- the curse of dimensionality,
- model building and evaluation
- on wine dataset
- using Python Scikit-learn package

KNN

- **Pros**
- The training phase of K-nearest neighbor classification is much faster compared to other classification algorithms.
- There is no need to train a model for generalization, That is why KNN is known as the simple and instance-based learning algorithm.
- KNN can be useful in case of nonlinear data.
- It can be used with the regression problem.
- Output value for the object is computed by the average of k closest neighbors value.

KNN

- **Cons**
- The testing phase of K-nearest neighbor classification is slower and costlier in terms of time and memory.
- It requires large memory for storing the entire training dataset for prediction.
- KNN requires scaling of data because KNN uses the Euclidean distance between two data points to find nearest neighbors.
- Euclidean distance is sensitive to magnitudes.
- The features with high magnitudes will weight more than features with low magnitudes.
- KNN also not suitable for large dimensional data.

Improving KNN

- For better results, normalizing data on the same scale is highly recommended.
- Generally, the normalization range considered between 0 and 1.
- KNN is not suitable for the large dimensional data.
- In such cases, dimension needs to reduce to improve the performance.
- Also, handling missing values will help us in improving results.

- Class review of Projects to date

References and Credits

- John Canny, CS 194 Fall 2015, UC Berkeley
- Datacamp